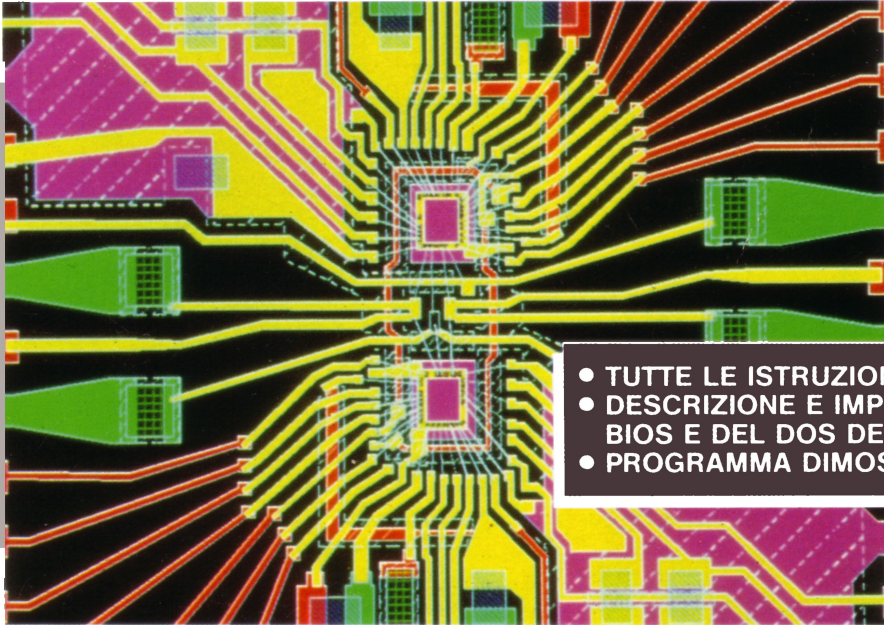
USO AVANZATO
DEL PCXT IBM

8086 - 8088
PROGRAMMAZIONE

SECONDA EDIZIONE

- TUTTE LE ISTRUZIONI
- DESCRIZIONE E IMPIEGO DEL BIOS E DEL DOS DEL PC IBM
- PROGRAMMA DIMOSTRATIVO

JAMES W. COFFRON

GRUPPO EDITORIALE
JACKSON

8086 - 8088
PROGRAMMAZIONE

JAMES W. COFFRON

GRUPPO
EDITORIALE
JACKSON
Via Rosellini, 12
20124 Milano

INDICE

PARTE 1

Concetti di programmazione in generale e dell'8086/8088 in particolare.

PARTE 2

Entra nelle applicazioni del mondo reale, mostrando come usare le informazioni sull'8088 per controllare il PC IBM.

APPENDICI

Sono inoltre incluse queste utili appendici:

INTRODUZIONE

Benvenuti nell'eccitante mondo dei microprocessori a 16- bit. Immaginati di programmare, controllare e usare uno dei piu' potenti e versatili microprocessori a 16-bit: l'8086. Questo microprocessore puo' accedere a oltre un milione di locazioni di memoria - enormemente di piu' delle 64000 disponibili nella maggioranza dei microprocessori a 8-bit. L'8086 è attualmente uno dei microprocessori a 16-bit più usati nell'industria. Ha un potentissimo set di istruzioni che permette di realizzare qualsiasi applicazione software. Ha, ad esempio, due istruzioni che permettono direttamente la moltiplicazione e la divisione di numeri.

Mentre s'impara come programmare l'8086, s'impara anche a programmare il microprocessore 8088. Questi due tipi di microprocessori hanno i set di istruzioni identici ed il software scritto per uno puo' essere utilizzato per l'altro senza cambiamenti.

Sia l'8086 che l'8088 sono adattati per la comunicazione con dispositivi e porte IO . Infatti la loro architettura puo' sopportare oltre 64000 porte singole di input e output. Queste porte possono essere qualsiasi combinazione di 8 o 16 bit. Inoltre l'8086 e l'8088 offrono un potente schema di interruzione che permette di far riferimento a qualsiasi indirizzo del sistema per le routine di servizio. Vi sono vettori di interruzione speciali per il single step e per gli errori interni, come la divisione per zero. Inoltre qualsiasi routine di servizio di un'interruzione puo' essere invocata via hardware o software.

*L'8086 e l'8088 sono stati progettati anche per facilitarne l'uso in applicazioni con sistemi di multiprocessori. Alla luce di questo l'INTEL CORPORATION ha progettato parecchi coprocessori da usare con l'8086*8088.*

I precedenti argomenti ed altri saranno trattati in questo testo. Si imparerà a programmare l'8086-8088 sia esternamente che internamente. Vi sarà fatto vedere come funziona, dall'architettura interna del microprocessore ai modi avanzati di indirizzamento. Ogni istruzione sarà descritta individualmente e saranno dati semplici esempi di veri e propri programmi; inoltre verranno presentati esempi di programmi reali scritti per il PC IBM che utilizza il microprocessore 8088.

Così, che siate principianti nella programmazione dei microprocessori in linguaggio assembly o che siate esperti programmatori, troverete qualcosa per voi in questo testo.

Dopo aver letto questo libro dovreste essere in grado di pro-

*grammare l'8086*8088 per ogni applicazione. Inoltre troverete che questo testo sarà per voi un preziosò riferimento anche successivamente.*

Quindi, iniziamo! Prendiamo la strada che portera' alla comprensione della potenza di uno fra i più avanzati microprocessori 16-bit attualmente in uso.

CAPITOLO 1

CONCETTI BASE

INTRODUZIONE

Iniziamo con una discussione dei concetti e delle definizioni di base usati nella programmazione dei calcolatori.

Chi ha già una certa familiarità con questi concetti può dare uno sguardo veloce ai contenuti di questo capitolo e passare quindi al Capitolo 2. Suggeriamo tuttavia che anche chi è un esperto programmatore legga tutto questo capitolo per familiarizzarsi con il tipo di approccio usato nel testo.

COS'È LA PROGRAMMAZIONE?

Quando la soluzione di un dato problema è espressa come una procedura passo per passo viene definita algoritmo. Un algoritmo può essere espresso in qualsiasi linguaggio o simbolismo; deve comunque poter essere eseguito in un numero finito di passi. Un semplice esempio di un algoritmo è quello per l'apertura di una porta chiusa a chiave:

1) inserire la chiave nella toppa

2) girare la chiave di un intero giro verso sinistra

3) afferrare la maniglia

4) girare la maniglia e spingere la porta.

A questo punto se l'algoritmo è corretto per il modello di serratura coinvolto, la porta verrà aperta.

Una volta espressa la soluzione del problema in forma di algoritmo, il passo successivo è quello di tradurla in un linguaggio comprensibile dal calcolatore. Attualmente solo un sottoinsieme ben definito di un linguaggio naturale — chiamato linguaggio di programmazione — è capito da un calcolatore. La conversione di un algoritmo in una sequenza di istruzioni appartenenti ad un linguaggio di programmazione è chiamata programmazione. La fase di traduzione vera e propria dell'algoritmo nel linguaggio di programmazione è chiamata codifica. La programmazione non comprende solo la codifica ma an-

Codifica di un programma

che il progetto completo dei programmi e della "struttura dati" che implementeranno l'algoritmo.

Una programmazione efficace richiede non solo una comprensione delle possibili tecniche di implementazione di algoritmi standard ma anche un uso intelligente delle risorse hardware del calcolatore (come registri interni, memoria e dispositivi periferici) e delle appropriate strutture dati. (NOTA: esamineremo queste tecniche nei capitolo seguenti).

La programmazione richiede inoltre una stretta disciplina di documentazione. Un programma ben documentato è sempre comprensibile non solo dall'autore, ma anche dagli altri utenti. La documentazione deve essere sia interna che esterna. La documentazione interna è rappresentata dai commenti usati in un programma per spiegare le operazioni che vengono eseguite. La documentazione esterna comprende i documenti del progetto che sono separati dal programma, come ad esempio spiegazioni scritte, manuali e diagrammi di flusso.

Documentazione interna ed esterna

Un passo intermedio è sempre presente fra la scrittura di un algoritmo e la creazione del programma. Durante questo passo, illustrato in dettaglio nel paragrafo seguente, vengono redatti i diagrammi di flusso.

DIAGRAMMI DI FLUSSO

Un diagramma di flusso è una rappresentazione simbolica di un algoritmo, abitualmente espressa in una sequenza di rettangoli e rombi.

Componenti fondamentali di un diagramma di flusso

In un diagramma di flusso, i rettangoli sono sempre usati per rappresentare comandi (o statement eseguibili) e i rombi sono usati per rappresentare selezioni come ad esempio "Se l'informazione X è vera, allora esegui l'azione A, se no esegui l'azione B». La Figura 1.1. mostra un esempio di diagramma di flusso. La stesura dei diagrammi di flusso è uno stadio intermedio, molto raccomandato, fra la specificazione di un algoritmo e l'effettiva codifica di una soluzione. È stato osservato che probabilmente solo il 10% dei programmatori è in grado di scrivere un programma con successo senza preparare prima un diagramma di flusso. Si è notato anche che, sfortunatamente, il 90% dei programmatori sono convinti di appartenere a quel 10% che può fare a meno dei diagrammi di flusso. Il risultato è che in media l'80% dei programmi falliscono quando si tenta di farli eseguire dal calcolatore. (Queste percentuali sono ovviamente piuttosto approssimative)

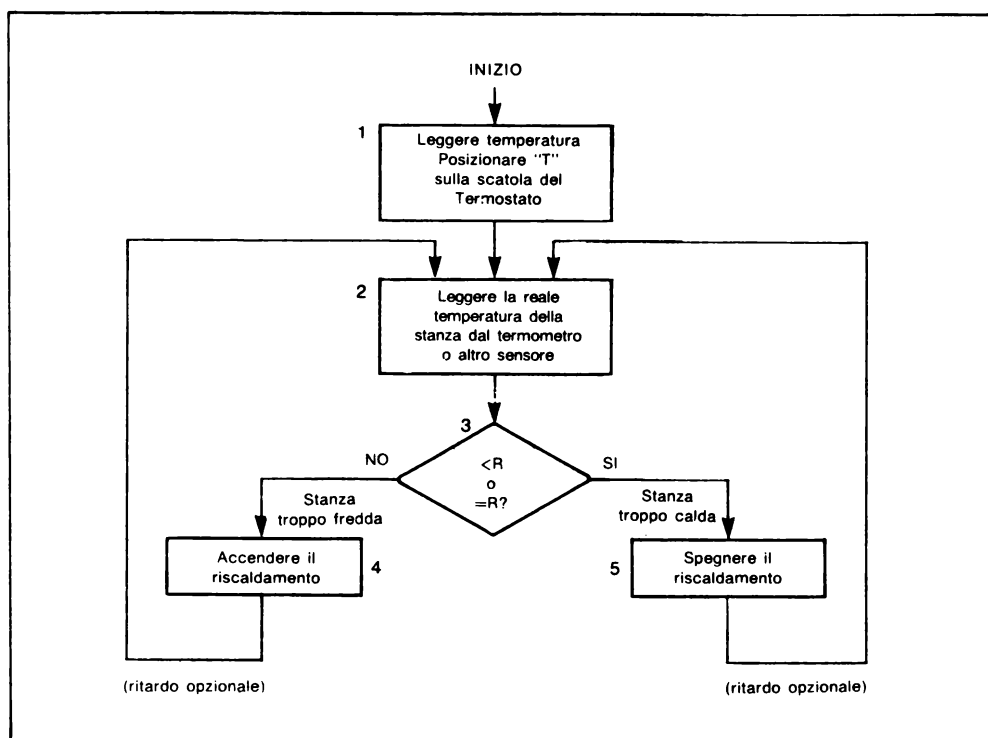



Figura 1.1 — Diagramma per tenere costante la temperatura di una stanza.

In breve, molti nuovi programmatori vedono raramente la necessità di scrivere un diagramma di flusso prima della codifica del programma, e questo risulta nella stesura di programmi sbagliati che successivamente richiedono molto tempo per il test e la correzione (fase di debugging).

L'uso dei diagrammi di flusso è, perciò, altamente raccomandato in ogni caso. La stesura dei diagrammi di flusso richiede generalmente una piccola aggiunta di tempo prima della codifica, ma di solito ne risulta un programma chiaro che viene eseguito correttamente in breve tempo. Una volta che la procedura di redazione dei diagrammi di flusso è ben compresa, una piccola percentuale di programmatori può eseguire questo stadio mentalmente, senza usare carta.

Sfortunatamente, in questi casi i programmi che essi scrivono sono di difficile comprensione per gli altri, dal momento che non è utilizzabile la documentazione normalmente fornita da un diagramma di flusso. Tutti raccomandano l'uso di diagrammi di flusso per ogni programma la cui lunghezza è superiore a 10 o 15 istruzioni. Molti esempi di diagrammi di flusso saranno forniti durante tutto il testo.

RAPPRESENTAZIONE DELL'INFORMAZIONE

Tutti i calcolatori manipolano le informazioni sotto forma di numeri o caratteri. Noi ora esamineremo la rappresentazione interna ed esterna dell'informazione in un calcolatore.

Rappresentazione interna dell'informazione

L'informazione è immagazzinata in un calcolatore come gruppi di bit. Un bit rappresenta una cifra binaria. A causa delle limitazioni dell'elettronica convenzionale, la maggior parte delle rappresentazioni pratiche dell'informazione usa la logica a due stati. I due stati usati nei circuiti dell'elettronica digitale sono generalmente "ACCESO" e "SPENTO" (on-off).

Questi stati sono rappresentati generalmente con i simboli 0 e 1. Poichè questi circuiti sono usati per implementare funzioni logiche, vengono chiamati circuiti logici binari. Come risultato, ogni elaborazione dell'informazione è attualmente eseguita in un formato binario. Un gruppo di 8 bit è chiamato byte. Un gruppo di 4 bit è chiamato nibble.

Byte e nibble

Vediamo ora come l'informazione è rappresentata internamente in questo formato binario. Due entità devono essere rappresentate in un calcolatore: il programma (che è una sequenza di istruzioni) e i dati su cui il programma opera.

I dati possono includere numeri o caratteri alfanumerici. Discutiamo ora la rappresentazione di istruzioni, numeri e caratteri alfanumerici nel formato binario. Nella maggior parte degli esempi che seguono si usano solo 8 bit per facilitare la spiegazione. Gli esempi possono comunque essere facilmente estesi oltre gli 8 bit.

Rappresentazione dei dati numerici

La rappresentazione di numeri in forma binaria non è un compito semplice: vari casi devono essere ben distinti. Dobbiamo essere capaci di rappresentare sia i numeri interi e sia i numeri con decimali. Occupiamoci ora di questi problemi e delle soluzioni possibili. Gli interi possono essere rappresentati usando una rappresentazione binaria diretta.

Rappresentazione binaria diretta

La rappresentazione binaria diretta è semplicemente la rappresentazione del valore in base 10 di un numero in un sistema binario. Nel sistema binario il bit all'estrema destra rappresenta 2 elevato a potenza zero. Il successivo bit verso sinistra

rappresenta 2 elevato alla potenza di 1, il successivo 2 a potenza 2, e il bit all'estrema sinistra 2 alla settima, che è uguale a 128. Per esempio:

$b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$

rappresenta

$b_7 \cdot 2^7 + b_6 \cdot 2^6 + b_5 \cdot 2^5 + b_4 \cdot 2^4 + b_3 \cdot 2^3 + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0$

Le potenze di 2 sono

$2^7 = 128,\ 2^6 = 64,\ 2^5 = 32,\ 2^4 = 16,\ 2^3 = 8,\ 2^2 = 4,\ 2^1 = 2,\ 2^0 = 1$

La rappresentazione binaria è analoga alla rappresentazione decimale dei numeri. Per esempio 123 in base 10 rappresenta:

$$
\begin{array}{rrrrr}
 & 1 \times & 100 & = & 100 \\
+ & 2 \times & 10 & = & 20 \\
+ & 3 \times & 1 & = & 3 \\
\hline
 & & & & 123
\end{array}
$$

Da notare che $100 = 10^2$, $10 = 10^1$, $1 = 10^0$. In questa notazione di tipo posizionale ogni cifra rappresenta una potenza di 10. Nel sistema binario ogni cifra binaria o bit rappresenta una potenza di 2, invece di una potenza di 10 come nel sistema decimale. Guardiamo un esempio di numero binario. 00001001 in binario rappresenta:

$$
\begin{array}{lrll}
1 \times & 1 & = 1 & (2^0) \\
0 \times & 2 & = 0 & (2^1) \\
0 \times & 4 & = 0 & (2^2) \\
1 \times & 8 & = 8 & (2^3) \\
0 \times & 16 & = 0 & (2^4) \\
0 \times & 32 & = 0 & (2^5) \\
0 \times & 64 & = 0 & (2^6) \\
0 \times & 128 & = 0 & (2^7) \\
\hline
 & & \multicolumn{2}{l}{\text{o } 9 \textit{ in decimale}}
\end{array}
$$

Guardiamo ora un altro esempio.
10000001 rappresenta:

1	×	1	=	1
0	×	2	=	0
0	×	4	=	0
0	×	8	=	0
0	×	16	=	0
0	×	32	=	0
0	×	64	=	0
1	×	128	=	128

o 129 *in decimale*

Perciò 10000001 rappresenta il numero decimale 129. L'esame della rappresentazione binaria dei numeri rende facile la comprensione del perchè i bit sono numerati da 0 a 7, andando da destra a sinistra. Il bit 0 è b0 e corrisponde a 2^0, il bit 1 è b1 e corrisponde a 2^1 e così di seguito.

Gli equivalenti binari dei numeri da 0 a 255 sono mostrati in Figura 1.2.

Da decimale a binario

Inversamente a ciò che abbiamo fatto ora possiamo computare l'equivalente binario del numero decimale 11:

11 ÷ 2 = 5 resto 1 > 1 *(bit di ordine basso)*
5 ÷ 2 = 2 resto 1 > 1
2 ÷ 2 = 1 resto 0 > 0
1 ÷ 2 = 0 resto 1 > 1 *(bit di ordine alto)*

Algoritmo di conversione da decimale a binario

Il numero binario equivalente è 1011 (si legga la colonna più a destra dal fondo verso l'alto). L'equivalente binario di un numero decimale può essere ottenuto dividendo continuamente per 2 finchè non si ottiene quoziente 0.

Operazioni su dati binari

Le regole aritmetiche per i numeri binari sono semplici.
Le regole per l'addizione sono:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = (1) 0

DECIMALE	BINARIO	DECIMALE	BINARIO
0	00000000	32	00100000
1	00000001	33	00100001
2	00000010	•	
3	00000011	•	
4	00000100	•	
5	00000101	63	00111111
6	00000110	64	01000000
7	00000111	65	01000001
8	00001000	•	
9	00001001	•	
10	00001010	•	
11	00001011	127	01111111
12	00001100	128	10000000
13	00001101	129	10000001
14	00001110		
15	00001111	•	
16	00010000	•	
17	00010001	•	
•			
•			
•		254	11111110
31	00011111	255	11111111

Figura 1.2 — Tabella Decimale-Binaria.

dove (1) rappresenta un "riporto" di 1 (notare che 10 è l'equivalente binario del 2 decimale). La sottrazione binaria può essere eseguita mediante l'addizione del complementare. Discuteremo sulla sottrazione binaria una volta imparato come rappresentare i numeri negativi.

Addizione binaria

Consideriamo ora un esempio che riguarda l'addizione.

```
 (2)      10
+(1)     +01
-----    -----
 (3)      11
```

L'addizione è eseguita come si fa con i numeri in base dieci: sommando le colonne partendo da destra e andando verso sinistra. Per prima cosa si somma la colonna più a destra:

```
  1 0
 +0 1
--------------------------------
    1   (0 + 1 = 1. Non c'è riporto)
```

poi la colonna successiva

```
  1 0
 +0 1
--------------------------------
  1 1   (1 + 0 = 1. Non c'è riporto)
```

Vediamo ora altri esempi di addizione binaria.

```
   0010 (2)        0011 (3)
 + 0001 (1)      + 0001 (1)
 ----------      ----------
   0011 (3)        0100 (4)
```

L'ultimo esempio evidenzia il ruolo del riporto.

Facciamo attenzione al bit più a destra: 1+1 = (1) 0. Un riporto di 1 è stato generato e deve quindi essere sommato al bit successivo:

```
    001   (la colonna 0 è già stata sommata)
 +  000
 +    1   (riporto)
 -------
 = (1)0   (dove (1) indica un nuovo riporto
           nella colonna 2)
```

Il risultato finale è 0100.
Consideriamo ora un altro esempio:

```
   0111       (7)
 + 0011     + (3)
 ----------------
   1010      (10)
```

Con 8 bit è perciò possibile rappresentare direttamente i numeri da 00000000 a 11111111 (cioè da 0 a 255). Due limitazioni comunque risaltano immediatamente. 1) stiamo rappresentando solo numeri positivi 2) la grandezza di questi numeri è limitata a 255, se usiamo solo 8 bit.

Parliamo dunque di queste limitazioni.

Numeri binari con bit di segno

In una rappresentazione con bit di segno, il bit all'estrema sinistra è usato per indicare il segno del numero. Tradizionalmente, 0 e' usato per indicare un numero positivo e 1 per indicare un numero negativo. Per esempio 11111111 rappresenta −127, mentre 01111111 rappresenta 127. Possiamo ora rappresentare i numeri positivi e negativi ma abbiamo ridotto la grandezza massima di questi numeri a 127. Vediamo un altro esempio: 00000001 rappresenta 1 (il primo 0 è un "+" seguito da 0000001 = 1) e 10000001 è = −1 (il primo 1 è "−").

Affrontiamo ora il problema grandezza.

Rappresentare numeri grandi richiede necessariamente un uso di un gran numero di bit. Ad esempio se usiamo 16 bit (2 byte) siamo in grado di specificare numeri da −32 K a +32 K usando la rappresentazione con bit di segno (1 K nel gergo informatico rappresenta 1024). Il bit 15 è usato per il segno e i rimanenti 15 bit (dal bit 14 al bit 0) sono usati per la grandezza: 2^{15} = 32 K.

Se vogliamo rappresentare interi grandi, è necessario usare internamente un grande numero di bit. Questo è il motivo per cui le versioni più semplici del BASIC, e altri linguaggi, forniscono una precisione limitata per gli interi.

Questo perché essi possono usare solo un formato intero corto per i numeri che manipoliamo.

Alcune versioni migliori del BASIC e altri linguaggi forniscono un maggior numero di cifre decimali significative ma di conseguenza consumano molti byte per ciascun numero.

Risolviamo un altro problema: l'efficienza in termini di velocità di esecuzione.

Compiamo un'addizione nella rappresentazione con bit di segno che abbiamo appena introdotto. Vogliamo addizionare +7 e −5

+ 7 è rappresentato con 00000111
– 5 è rappresentato con 10000101

---

la somma binaria è 10001100, o –12

Questo risultato non è corretto. Il risultato esatto è +2. Abbiamo trascurato il fatto che per usare questa rappresentazione, devono essere compiute azioni particolari in dipendenza dal segno.

Questo si trasforma in un aumento delle complessità e in una riduzione delle prestazioni. In altre parole l'addizione binaria di numeri con bit di segno non è fatta "lavorando correttamente". Questo è di disturbo. Chiaramente il calcolatore non deve solo rappresentare l'informazione, ma deve anche poter eseguire su di essa delle operazioni aritmetiche. La soluzione a questo problema è chiamata "rappresentazione in complemento a 2". Noi useremo la rappresentazione in complemento a 2 invece della rappresentazione con bit di segno. Prima di introdurre il complemento a 2, introduciamo uno stadio intermedio: il complemento a 1.

Complemento a 1

Nella rappresentazione in complemento a 1 tutti i numeri interi positivi sono rappresentati nel loro formato binario corretto. Per esempio +3 è rappresentato come al solito con 00000011. Tuttavia il suo complementare –3 è ottenuto complementando tutti i bit presenti nella rappresentazione originale. Nella complementazione ogni 0 è trasformato in 1 e ogni 1 in 0. Nel nostro esempio la rappresentazione in complemento a 1 di –3 è 11111100. Vediamo un altro esempio:

+ 2 è 00000010
– 2 è 11111101

È da notare che con questa rappresentazione i numeri positivi cominciano con uno 0 a sinistra e i negativi con un 1 a sinistra.

Per prova addizioniamo –4 e +6:

–4 è 11111011
+6 è 00000110

---

la somma è: (1) 00000001

dove (1) indica il riporto (carry). Il risultato corretto dovrebbe essere 2 o 00000010.

Riproviamo con un altro esempio:

–3 è	11111100
–2 è	11111101
la somma è (1)	11111001

o –6 più un riporto. Il risultato corretto è –5. La rappresentazione di –5 è 11111010. Non ha funzionato.

Questa rappresentazione rappresenta numeri positivi e negativi. Tuttavia il risultato di un'addizione eseguita secondo le regole ordinarie non dà sempre un esito corretto. Useremo ora un'altra rappresentazione, derivata dalla rappresentazione in complemento a 1 e chiamata rappresentazione in complemento a 2.

Rappresentazione in complemento a 2

Calcolo del complemento a due di un numero

Nella rappresentazione in complemento a 2 i numeri positivi sono rappresentati, come di solito, col bit di segno, proprio come nel complemento a 1. La differenza sta nella rappresentazione dei numeri negativi. Un numero negativo rappresentato in complemento a 2 è ottenuto eseguendo dapprima il complemento a 1 e quindi aggiungendo 1.

Esempio: +3 è rappresentato in binario segnato da 00000011. La rappresentazione di –3 in complemento a 1 è 11111100. La rappresentazione in complemento a 2 è ottenuta aggiungendo 1, ed è 11111101.

Facciamo un'addizione:

(3)	00000011
+(5)	+00000101
(8)	00001000

Il risultato è corretto.

Facciamo ora una sottrazione:

(3)	00000011
(–5)	+11111011
	11111110

Ora identifichiamo il risultato eseguendo il complemento a 2:

(il complemento a 1 di 11111110 è)	00000001
(sommando 1)	+ 1
(quindi nella rappresentazione in complemento a 2 è)	00000010 (o +2)

Il risultato +11111110 rappresenta −2. È corretto.

Abbiamo provato l'addizione e la sottrazione ed i risultati sono corretti (ignorando il riporto). Sembra quindi che il complemento a 2 funzioni. Vogliamo sommare +4 e −3 (la sottrazione è computata sommando il complemento a 2)

+4 è	00000100
−3 è	11111101
il risultato è (1)	00000001

Se ignoriamo il riporto il risultato è 00000001 (cioè 1 in decimale). Questo risultato è corretto. Anche senza dare l'intera dimostrazione matematica possiamo affermare che questa rappresentazione funziona.

Operazioni con numeri in complemento a due

Nel complemento a 2 è possibile sommare o sottrarre numeri con segno senza curarsi del segno stesso. Usando le regole abituali dell'addizione binaria il risultato è corretto, incluso il segno. Il riporto è ignorato. Questo è un importantissimo vantaggio. Se così non fosse, noi dovremmo ogni istante correggere il risultato per via del segno, causando un grande rallentamento nel tempo di addizione e sottrazione. Per amore della completezza, possiamo affermare che il complemento a 2 è la più conveniente rappresentazione che viene usata per i processori più semplici come i microprocessori.

Su processori più complessi possono essere usate altre rappresentazioni. Per esempio può essere usata la rappresentazione in complemento a 1, però se viene usata ci vuole un circuito speciale per correggere il risultato.

Da qui in avanti, ogni intero con segno sarà rappresentato implicitamente nella notazione in complemento a 2. La Figura 1.3 riporta una tabella di numeri espressi in complemento a 2. Daremo ora altri esempi che illustrano le regole della rappresentazione in complemento a 2. In particolare C denota una possibile condizione in cui si verifica un riporto (carry) o un prestito (C è il bit 8 del risultato); V indica un overflow del complemento a 2 e rivela quando il segno del risultato è cambiato accidentalmente perchè i numeri sono troppo grandi.

+	CODICE DEL COMPLEMENTO A DUE	−	CODICE DEL COMPLEMENTO A DUE
+ 127	01111111	− 128	10000000
+ 126	01111110	− 127	10000001
+ 125	01111101	− 126	10000010
...		− 125	10000011
		...	
+ 65	01000001	− 65	10111111
+ 64	01000000	− 64	11000000
+ 63	00111111	− 63	11000001
...		...	
+ 33	00100001	− 33	11011111
+ 32	00100000	− 32	11100000
+ 31	00011111	− 31	11100001
...		...	
+ 17	00010001	− 17	11101111
+ 16	00010000	− 16	11110000
+ 15	00001111	− 15	11110001
+ 14	00001110	− 14	11110010
+ 13	00001101	− 13	11110011
+ 12	00001100	− 12	11110100
+ 11	00001011	− 11	11110101
+ 10	00001010	− 10	11110110
+ 9	00001001	− 9	11110111
+ 8	00001000	− 8	11111000
+ 7	00000111	− 7	11111001
+ 6	00000110	− 6	11111010
+ 5	00000101	− 5	11111011
+ 4	00000100	− 4	11111100
+ 3	00000011	− 3	11111101
+ 2	00000010	− 2	11111110
+ 1	00000001	− 1	11111111
+ 0	00000000		

Figura 1.3 — Tabella del complemento a due.

L'overflow è essenzialmente un riporto interno dal bit6 al bit7 (il bit di segno). Questo fatto verrà chiarito in seguito. Dimostriamo ora il ruolo del riporto C e dell'overflow V.

Il riporto C

Eccone un esempio:

(128)	10000000
+ (129)	+ 10000001
(257)=(1)	00000001

dove (1) indica un riporto. Il risultato richiede un nono bit (il bit8, poichè quello più a destra è bit0). Questo è il bit del riporto (o carry bit). Se assumiamo che il riporto è il nono bit del risultato, possiamo riconoscere il risultato come il numero binario 100000001 = 257. Tuttavia il riporto deve essere riconosciuto e maneggiato con cura. Dentro il microprocessore del nostro esempio, i registri che contengono le informazioni sono larghi solo 8 bit. Una volta immagazzinato il risultato solo i bit da 0 a 7 possono essere conservati. Un riporto, perciò, richiede sempre un processo speciale: deve essere rilevato con opportune istruzioni e deve essere trattato in modo particolare. Bisogna cioè stabilire se immagazzinarlo in qualche parte (con un'istruzione speciale) o ignorarlo o decidere che è un errore (se il maggior risultato consentito è 11111111).

L'overflow

Eccone un esempio:

È stato generato un riporto interno dal bit6 al bit7. Questo riporto interno viene chiamato overflow. Il risultato è ora negativo "per caso". Questa situazione deve essere scoperta e quindi corretta. Esaminiamo un'altra situazione:

```
     11111111      (−1)
    +11111111     +(−1)
-------------------------
(1) 11111110       (−2)
 ↓
riporto
```

In questo caso viene generato un riporto interno dal bit6 al bit7, e inoltre dal bit7 al bit C. Le regole dell'aritmetica in complemento a 2 specificano che questo vettore riporto può essere ignorato. Il risultato quindi è corretto. Questo perchè il riporto dal bit6 al bit7 non fa cambiare il bit del segno.

Il riporto da bit6 a bit7 non è una condizione di overflow. Quando si opera con numeri negativi l'overflow non è semplicemente un riporto dal bit6 al bit7. Esaminiamo ancora un esempio:

```
     11000000      (−64)
    +10111111      (−65)
-------------------------
(1) 01111111      (+127)
 ↓
riporto
```

Questa volta non c'è stato un riporto interno dal bit6 al bit7 ma c'è stato un riporto esterno. Il risultato è scorretto dato che il bit7 è stato cambiato. Questa potrebbe essere l'indicazione di un overflow.

L'overflow avviene in 4 situazioni:

Cause che determinano l'overflow

1) L'addizione di numeri positivi troppo grandi.
2) L'addizione di numeri negativi troppo grandi (in valore assoluto).
3) La sottrazione di un numero positivo grande da un numero negativo grande.
4) La sottrazione di un numero negativo grande da un numero positivo grande.

Rivelazione di un overflow

Cerchiamo ora di migliorare la nostra definizione di overflow: tecnicamente l'indicatore di overflow è uno speciale bit riservato per questo scopo, chiamato codice di condizione, che sarà posto a 1 quando c'è un riporto dal bit6 al bit7 e questo riporto non è esterno. Il bit di overflow verrà inoltre posto a 1 quando non c'è riporto dal bit6 al bit7 e c'è un riporto esterno.

Questo indica che il bit7 (il segno del risultato) è stato accidentalmente cambiato. Per il lettore tecnicamente esperto diciamo che il bit di overflow è posto a 1 o azzerato in base al risultato dell'esecuzione di un or esclusivo tra il riporto interno

e il riporto esterno del bit 7 (il bit di segno). Praticamente ogni microprocessore è provvisto di uno speciale flag di overflow per rilevare automaticamente questa condizione (condizione che richiede un'azione correttiva). L'overflow indica che il risultato di una sottrazione o di un'addizione richiede più bit di quelli presenti nel registro standard di 8 bit usato per contenere il risultato.

Significato dell'overflow

Il riporto e l'overflow

Il bit del riporto e quello dell'overflow sono chiamati codici di condizione e sono contenuti in ogni microprocessore. Impareremo il loro uso per una efficace programmazione nel Capitolo 2. Questi due indicatori sono posti in un registro speciale chiamato "registro dei flag" o "registro di stato".

Questo registro inoltre contiene degli indicatori addizionali (descritti nel Capitolo 4).

Esempi. Vedremo ora degli esempi reali che illustrano il riporto e l'overflow. In ogni esempio, il simbolo V indica l'overflow e C il riporto. Se non c'è stato overflow V sarà 0, se invece c'è stato un overflow allora V sarà =1 (lo stesso vale per C). Ricordiamo inoltre che le regole del complemento a 2 specificano che il riporto deve essere ignorato (in questa sede la dimostrazione matematica non viene fornita). Esaminiamo i seguenti esempi:

Positivo-positivo

```
        00000110      (+6)
       +00001000      (+8)
   ---------------------------
        00001110     (+14)   V=0   C=0
     (CORRETTO)
```

Positivo-positivo con overflow

```
        01111111    (+127)
       +00000001      (+1)
   ---------------------------
        10000000    (-128)   V=1   C=0
       (ERRORE)
```

Il sopracitato esempio viene invalidato dato che si è verificato un overflow.

Positivo-negativo (risultato positivo)

00000100	(+4)		
+11111110	(−2)		
(1) 00000010	(+2)	V=0	C=1 (Trascurabile)
(CORRETTO)			

Positivo-negativo (risultato negativo)

00000010	(+2)		
+11111100	(−4)		
11111110	(−2)	V=0	C=0
(CORRETTO)			

Negativo-negativo

11111110	(−2)		
+11111100	(−4)		
(1)11111010	(−6)	V=0	C=1 (Trascurabile)
(CORRETTO)			

Negativo-negativo con overflow

10000001	(−127)		
+11000010	(−62)		
(1)01000011	(+67)	V=1	C=1
(ERRORE)			

Nell'ultimo esempio è avvenuto un "underflow" dovuto all'addizione di 2 numeri negativi troppo grandi. Infatti il risultato −189 è in valore assoluto troppo grande per poter essere contenuto in 8 bit.

Rappresentazione in formato fisso

Ora sappiamo come rappresentare gli interi segnati, però non abbiamo ancora risolto il problema della grandezza. Se vogliamo rappresentare degli interi grandi, avremo bisogno di parecchi byte. Per eseguire bene è con efficienza operazioni aritmetiche è necessario usare un numero fisso di byte, anziché un numero variabile. Perciò una volta scelto un numero di byte rimane fissata la massima grandezza del numero che può essere rappresentato.

Il problema della "grandezza"

Troncamento del risultato

Consideriamo il seguente punto importante: il numero di bit, n, scelto per la rappresentazione in complemento a 2 è generalmente fisso per un certo programma. Se qualche risultato o computazione intermedia genera un numero che richiede più di n bit, qualche bit verrà perso. Il programma trattiene di solito i 7 bit più a sinistra (i più significativi) e lascia perdere i rimanenti.

Questo è chiamato "troncamento del risultato". Prendiamo in considerazione un esempio nel sistema decimale usando una rappresentazione a 6 cifre:

```
        123456
  ×        1,2
  ------------
        246912
       123456
  ------------
      148147,2
```

Il risultato richiede 7 cifre, il 2 dopo la virgola verrà perso, il risultato finale sarà 148147. È stato troncato.

Di solito questo metodo, fino a che la posizione della virgola non è persa, viene usato per ampliare il margine delle operazioni che possono essere fatte a scapito della precisione. Il problema è lo stesso con numeri binari. Questa rappresentazione in formato fisso, può causare una perdita di precisione ma può essere sufficiente per il calcolo o le operazioni matematiche. Sfortunatamente, nel caso della contabilità non è tollerabile la perdita di precisione. Per esempio, se un cliente batte sul registro di cassa un grande totale, potrebbe non essere accettabile avere un totale a 5 cifre che dovrebbe essere approssimato al dollaro. Perciò, tutte le volte che la precisione nel risultato è essenziale deve essere usata un'altra rappresentazione. La soluzione normalmente usata è la codifica BCD (BCD è l'acronimo di Binary Coded Decimal).

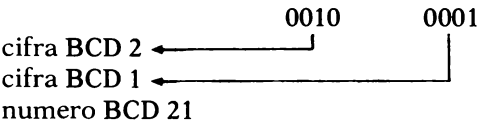
Rappresentazione BCD

Il principio usato nella rappresentazione numerica BCD è quello di codificare ciascuna cifra separatamente e usare tutti i bit che sono necessari per rappresentare esattamente l'intero numero.

Per codificare ogni cifra da 0 a 9 sono necessari 4 bit (3 bit forniscono solo 8 combinazioni e perciò non possono codificare 10 cifre - 4 bit forniscono 16 combinazioni e sono, per que-

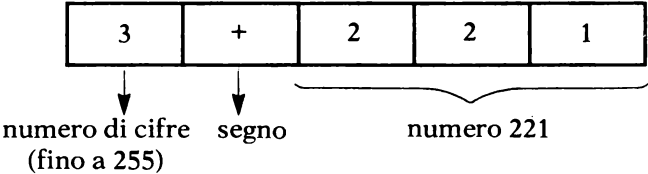
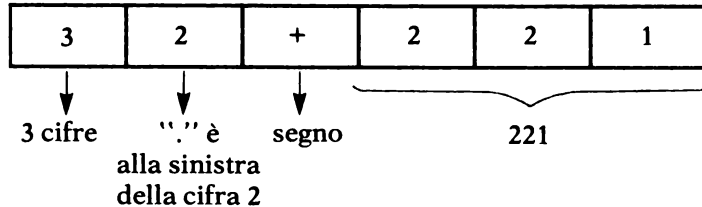
sto, sufficienti a codificare le cifre da 0 a 9). Si potrebbe notare, inoltre, che 6 dei possibili codici non saranno usati nella rappresentazione BCD (vedi Figura 1.4).

Rappresentazione BCD impaccata

Questo potrebbe diventare un problema quando si eseguono addizioni e sottrazioni. Dato che solo 4 bit sono necessari alla codifica di una cifra BCD, 2 cifre BCD possono essere codificate in ogni byte. Questo si chiama BCD impaccato. Per esempio 00000000 è 00 in BCD. 10011001 è 99 . Un codice BCD viene letto come segue:

Per rappresentare tutte le cifre BCD saranno usati tanti byte quanti sono necessari. Tipicamente uno o più nibble (semibyte) saranno usati all'inizio della rappresentazione per indicare il numero totale di nibble (cioè di cifre BCD usate). Un altro nibble o un byte sarà usato per indicare la posizione della virgola.

CODICE	SIMBOLO BCD	CODICE	SIMBOLO BCD
0000	0	1000	8
0001	1	1001	9
0010	2	1010	NON USATO
0011	3	1011	NON USATO
0100	4	1100	NON USATO
0101	5	1101	NON USATO
0110	6	1110	NON USATO
0111	7	1111	NON USATO

Figura 1.4 — Tabella - BCD.

Tuttavia le convenzioni possono variare. Ecco un esempio di rappresentazione di interi BCD multi-byte.

Questo numero rappresenta +221 (il segno può essere rappresentato con 0000 per indicare + e 0001 per indicare –, per esempio). In BCD si possono facilmente esprimere anche i numeri decimali. Per esempio +2.21 può essere rappresentato con:

Pro e contro della rappresentazione BCD

Il vantaggio del BCD è che produce un risultato assolutamente perfetto. Il suo svantaggio è che usa una grande quantità di memoria e risulta lento nelle operazioni aritmetiche. Ciò è accettabile nel campo della contabilità ma lo rende praticamente inutilizzabile negli altri casi.

Abbiamo ora risolto i problemi associati alla rappresentazione degli interi, interi segnati, e degli interi grandi. Abbiamo presentato un possibile metodo di rappresentare dei numeri decimali con la codifica BCD.

Esaminiamo ora il problema di rappresentare i numeri decimali in un formato di lunghezza fissa.

Rappresentazione in virgola mobile

Normalizzazione di un numero

Il principio alla base della rappresentazione in virgola mobile è che i numeri decimali sono rappresentati con un formato di lunghezza fissa. Per non sprecare bit la rappresentazione normalizza tutti i numeri. Per esempio: 0,000123 spreca 3 zeri a sinistra prima della cifra non nulla. Questi zeri non hanno significato se non per indicare la posizione della virgola decimale (indicata con un punto per uniformità con la rappresentazione adottata negli elaboratori). Normalizzando questo numero ne risulta $.123 \times 10^{-3}$, .123 è la mantissa normalizzata; –3 è l'esponente. Abbiamo normalizzato questo numero mediante l'eliminazione degli zeri non significativi prima della prima cifra non nulla e l'assegnamento di un opportuno valore all'esponente. Consideriamo un altro esempio:

Esempio: 22.1 è normalizzato come $.221 \times 10^2$. La forma generale della rappresentazione in virgola mobile è $M \times 10^e$, dove M è la mantissa e "e" l'esponente.

Può essere facilmente notato che un numero normalizzato è caratterizzato da una mantissa minore di 1 e maggiore o uguale a 0.1 in tutti i casi in cui il numero non è nullo. In altre parole questo può essere rappresentato matematicamente con:

$$0.1 < M < 1 \text{ o } 10^{-1} < M < 10^{0}$$

Analogamente nella rappresentazione binaria

$$2^{-1} < M < 2^{0} \text{ (o } 0.5 < M < 1)$$

Dove M è il valore assoluto della mantissa (trascurando il segno).

Per esempio:

111.01 è normalizzato come $.11101 \times 2^{3}$.

La mantissa è 11101 e l'esponente è 3.

Esempio pratico di rappresentazione in virgola mobile

Ora che abbiamo definito il principio della rappresentazione, esaminiamone il reale formato. Una tipica rappresentazione in virgola mobile appare in Figura 1.5. Sono usati 4 byte cioè 32 bit. Il primo byte a sinistra nella figura è usato per rappresentare l'esponente. Sia l'esponente che la mantissa saranno rappresentati in complemento a 2. Come risultato il massimo esponente sarà —128. "S" in Figura 1.5 indica il bit del segno. 3 byte sono usati per rappresentare la mantissa. Poiche' il primo bit nella rappresentazione in complemento a 2 indica il segno, ne rimangono 23 per rappresentare la grandezza della mantissa. Questo è solo un esempio di rappresentazione in virgola mobile. È possibile usare solo 3 byte ed è altrettanto possibile usarne più di 4.

La rappresentazione a 4 byte proposta sopra è di uso comune e rappresenta un ragionevole compromesso in termini di accuratezza, grandezza dei numeri, utilizzazione della memoria ed efficienza nelle operazioni aritmetiche.

Abbiamo ora esplorato i problemi associati alla rappresentazione dei numeri e abbiamo imparato a rappresentarli in forma intera con un segno o in forma decimale. Passiamo ora ad esaminare come sono rappresentati internamente i dati alfanumerici.

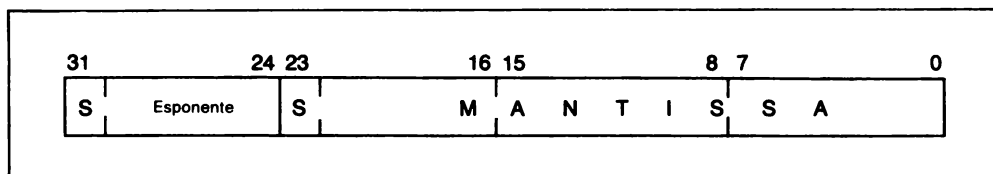



Figura 1.5 — Tipica rappresentazione in virgola mobile.

Rappresentazione dei dati alfanumerici

La rappresentazione dei dati alfanumerici (cioè caratteri) è molto semplice: tutti i caratteri sono codificati con un codice a 8 bit.

In generale nei calcolatori si usano solo 2 codici: l'ASCII e l'EBCDIC. ASCII sta per "American Standard Code for Information Interchange" ed è universalmente usato nei microprocessori. L'EBCDIC è una variazione dell'ASCII usata dall'IBM e per questo non è usato nei microcomputer a meno che non vengano interfacciati con un terminale IBM.

Esaminiamo brevemente la codifica ASCII. Codifichiamo 26 lettere maiuscole dell'alfabeto e 26 minuscole più 10 simboli numerici e circa 20 simboli speciali addizionali. Tutto questo è facilmente ottenibile con 7 bit che permettono 128 possibili codici. (Vedi Figura 1.6) Tutti i caratteri sono quindi codificati in 7 bit. L'ottavo bit, quando si usa, è il bit parità.

Verifica di parità

La parità è una tecnica per verificare che il contenuto di un byte non sia cambiato accidentalmente. Il numero di 1 presenti nel byte viene contato e l'ottavo bit viene posto a 1 nel caso di conto dispari per ottenere un totale pari. Questa operazione è chiamata parità.

La parità dispari (scrivendo l'ottavo bit, quello più a sinistra, in modo tale che il numero totale di 1 nel byte sia dispari) può essere anch'essa usata. Come esempio calcoliamo il bit parità di 0010011 usando la parità pari.

Il numero di 1 è 3 . Il bit di parità deve perciò essere 1, in modo che il numero totale dei bit a 1 sia 4 cioè pari. Il risultato è 10010011 dove il primo 1 è il bit di parità e 0010011 identifica il carattere. La tabella del codice ASCII a 7 bit è mostrata in Figura 1.6; in pratica questa tabella viene usata tal quale senza il bit di parità aggiungendo uno zero nella posizione più a sinistra, oppure col controllo di parità assegnando l'appropriato valore al bit sulla sinistra (il bit di parità). In situazioni specialistiche come nelle telecomunicazioni, possono essere usati altri codici come il codice a correzione d'errore. Tuttavia, descrizioni di queste codifiche vanno al di là dello scopo di questo testo.

Ora che abbiamo esaminato le rappresentazioni abituali sia per programmi che per dati interni al calcolatore, esaminiamo le possibili rappresentazioni esterne.

ESA	MSD	0	1	2	3	4	5	6	7
LSD	BITS	000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SPACE	0	@	P	_	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	- -
D	1101	CR	GS	—	=	M	]	m	}
E	1110	SO	RS	.	>	N	^	n	∿
F	1111	SI	US	/	?	O		o	DEL

Figura 1.6 — Tabella di conversione ASCII (vedi App. B per le abbreviazioni).

Rappresentazione esterna dell'informazione

La rappresentazione esterna dell'informazione si riferisce al modo in cui l'informazione è presentata all'utente (che generalmente è il programmatore). L'informazione viene presentata esternamente essenzialmente in 3 formati: binario, ottale o esadecimale, e simbolico. Esaminiamo questi formati.

1 - Binario: abbiamo visto che l'informazione è memorizzata internamente in byte che sono una sequenza di 8 bit (0 o 1). Talvolta è desiderabile mostrare questa informazione interna direttamente nel suo formato binario. Questa è conosciuta come rappresentazione binaria. Un semplice esempio è quello in cui l'informazione è rappresentata mediante dei LED, che sono essenzialmente lampadine in miniatura su un pannello frontale di un microcomputer. Nel caso di un microprocessore

BINARIO	OTTALE
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Figura 1.7 — Simboli Ottali.

ad 8 bit, il pannello frontale è equipaggiato tipicamente con 8 LED per esporre i contenuti di un qualunque registro interno. Un LED acceso indica 1, spento indica 0. Una tale rappresentazione binaria può essere usata per la ricerca accurata degli errori di un programma complesso, specialmente se questo implica input e output, ma è naturalmente poco pratica a livello umano. Nella maggior parte dei casi è più facile guardare un'informazione in forma simbolica. Per esempio 9 è più facile da comprendere e da ricordare che 1001. Per questo sono state progettate rappresentazioni più convenienti che migliorano lo scambio fra uomini e macchine.

2 - Ottale e esadecimale: l'ottale e l'esadecimale codificano in un unico simbolo rispettivamente 3 e 4 bit binari. L'ottale è un formato che usa 3 bit, dove ogni combinazione di 3 bit è rappresentata con un simbolo compreso fra 0 e 7 (vedi Figura 1.7.). Per esempio 00100100 binario è rappresentato come 044 in ottale. Un altro esempio: 11111111 è 377 in ottale. Viceversa, l'ottale 211 rappresenta 010001001 in binario. L'ottale veniva usato tradizionalmente nei più vecchi calcolatori che impiegavano svariati bit, da 8 a 64. Più recentemente con la predominanza dei microprocessori a 8 bit, il formato 8 bit è diventato lo standard e viene usata un'altra rappresentazione più pratica: l'esadecimale. Nella rappresentazione esadecimale un gruppo di 4 bit è codificato come una cifra esadecimale. Le cifre esadecimali sono rappresentate con simboli da 0 a 9 e con le lettere A, B, C, D, E, F. Per esempio 0000 è rappresentato

DECIMALE	BINARIO	ESADECIMALE	OTTALE
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	8	10
9	1001	9	11
10	1010	A	12
11	1011	B	13
12	1100	C	14
13	1101	D	15
14	1110	E	16
15	1111	F	17

Figura 1.8 — Codici Esadecimali.

con 0; 0001 con 1 e 1111 è rappresentato con la lettera F (vedi Figura 1.8).

Per esempio 10100001 in binario è rappresentato con A1 in esadecimale. L'esadecimale offre il vantaggio di codificare 8 bit in due cifre. Quindi è più facile da visualizzare e memorizzare ed è più veloce da scrivere nel calcolatore del suo equivalente binario. Perciò, per la maggior parte dei microprocessori l'esadecimale è il metodo preferito per la rappresentazione di gruppi di bit.

Tutte le volte che l'informazione presente nella memoria ha un significato, come nel caso della rappresentazione di testi o numeri, l'esadecimale non è conveniente per rappresentare il significato di queste informazioni per un utente umano.

Rappresentazione simbolica

La rappresentazione simbolica si riferisce alla rappresentazione esterna dell'informazione nella sua reale forma simboli-

ca. Per esempio i numeri decimali sono rappresentati come numeri decimali e non come sequenza di simboli esadecimali o bit. In modo analogo un testo è rappresentato tale e quale.

Naturalmente la rappresentazione simbolica è più pratica per l'utente. È usata tutte le volte che è disponibile un appropriato mezzo di esposizione, ad esempio un display CRT o una stampante (un CRT è uno schermo di tipo televisivo per visualizzare testi o grafici). Sfortunatamente nei sistemi più piccoli come una tastiera microcomputer, simili dispositivi sono antieconomici e l'utente è costretto alla comunicazione esadecimale col calcolatore.

Sommario delle rappresentazioni esterne

La rappresentazione simbolica di un'informazione è la più desiderabile, dato che è la più naturale per un utente umano. D'altra parte ciò richiede un costo elevato dovuto alla tastiera alfanumerica e ad una stampante o un display CRT. Quindi questo tipo di rappresentazione è difficilmente disponibile nei sistemi meno costosi. Un tipo alternativo di rappresentazione viene perciò usato e in tali casi l'esadecimale è la rappresentazione predominante. Solo in rari casi, come ad esempio nelle operazioni di debugging di precisione a livello di software ed hardware, viene usata la rappresentazione binaria. In questo caso i contenuti dei registri di memoria sono mostrati direttamente in formato binario.

Ora abbiamo visto come l'informazione è rappresentata internamente ed esternamente passiamo ad esaminare il microprocessore che manipola queste informazioni.

CAPITOLO 2

ALL'INTERNO DELL'8086/8088

INTRODUZIONE

In questo capitolo discuteremo l'architettura interna dei microprocessori 8086 e 8088. Cominceremo con un'ampia discussione dell'architettura di un microprocessore in generale. Verranno discussi importanti concetti comuni alla maggior parte dei microprocessori a singolo chip. Esamineremo poi il ciclo istruzione di un tipico microprocessore. Alla fine di questa discussione avrete una buona comprensione generale dell'organizzazione interna di un microprocessore e dell'esecuzione delle istruzioni.

Passeremo quindi all'esame dell'organizzazione interna dell'8086/8088. Vedremo da vicino le numerose caratteristiche di cui sono provvisti questi due microprocessori e le confronteremo con le caratteristiche tipiche degli altri microprocessori. Quando paragoneremo i processori, risulterà chiaro che i progettisti dell'8086/8088 hanno utilizzato nuove soluzioni per vecchi problemi ottenendo come risultato un potentissimo microprocessore di facile uso.

ARCHITETTURA GENERALE DI UN SISTEMA BASATO SU MICROPROCESSORE

Incominciamo la nostra discussione esaminando l'architettura hardware generale di un tipico sistema basato su microprocessore. Sebbene non avremo tempo per spiegare come questa architettura venga realizzata con l'8086/8088, mostreremo, per aiutarvi a comprendere meglio l'organizzazione interna di una CPU, come un microprocessore venga adattato nell'ambiente di un sistema. La Figura 2.1 mostra un diagramma a blocchi di un tipico sistema controllato da un microprocessore. Sulla sinistra del diagramma appare l'unità microprocessore (l'MPU) — in questo caso l'8086/8088 — che implementa le funzioni dell'unità centrale di elaborazione (la CPU) su un singolo chip. La CPU include un'unità logico-aritmetica (l'ALU), i suoi registri interni e l'unità di controllo (CU) che decodifica e sequenzializza internamente le istruzioni. (Illustreremo me-

Componenti della CPU: ALU, CU, registri

glio la CPU più avanti in questo capitolo). L'MPU ha tre bus: uno per gli indirizzi, uno per i dati e uno di controllo. Un bus è una collezione di segnali elettronici che derivano da una sorgente comune ed eseguono una funzione comune. Vediamo in Figura 2.1 che a fianco della CPU ci sono ROM, RAM e I/O. Questi sono i blocchi principali di un sistema microprocessore. Non importa quanto sia complesso il sistema o quali siano le sue funzioni, il diagramma a blocchi 2.1 lo descrive comunque accuratamente.

ROM: definizione e funzioni

Passiamo ora ad esaminare brevemente ciascuno dei principali blocchi, chiamati ROM, RAM e I/O. La ROM (o memoria di sola lettura) contiene il programma del sistema. Il vantaggio della memoria ROM è che i suoi contenuti sono permanenti: non si cancellano spegnendo il sistema. Per questa ragione la ROM contiene solitamente il bootstrap, o programma di controllo, che permette l'inizializzazione del sistema; inapplicazioni in cui i processi sono controllati da un calcolatore, quasi tutti i programmi risiedono in ROM, questo perchè di rado vengono cambiati e possono essere protetti da cali di ten-

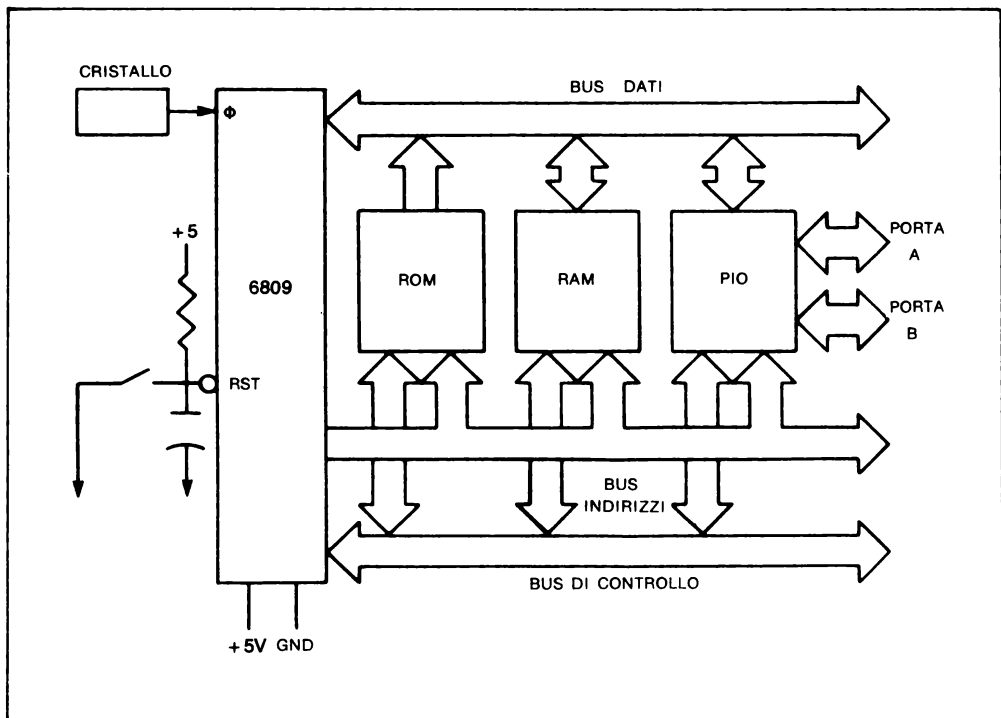



Figura 2.1 — Il diagramma mostra la tipica architettura di un sistema basato su microprocessore che usa ROM, RAM e I/O.

RAM: definizione e funzioni

sione. La RAM (o memoria ad accesso casuale) è la parte di memoria di lettura e scrittura del sistema.

In ambienti di sviluppo di programmi la maggior parte dei programmi risiede in RAM, al fine di poter essere facilmente modificati. Tali programmi possono essere tenuti nella RAM o trasferiti nella ROM quando ciò risulta conveniente. La RAM è volatile, quindi le informazioni si perdono spegnendo il sistema. In un sistema di controllo lo spazio della RAM è tipicamente piccolo (solo per i dati), viceversa in ambiente di sviluppo di programmi, lo spazio della RAM è generalmente grande perchè contiene i programmi e in più il software di sviluppo. Prima dell'uso i contenuti della RAM devono essere caricati da un dispositivo esterno.

Infine, il sistema contiene uno o più chip di interfaccia. Un chip di interfaccia permette la comunicazione fra il sistema e il mondo esterno. Il PIO è un chip di interfaccia frequentemente usato. PIO è l'acronimo di Parallel Input Output (input output paralleli). I PIO, come gli altri chips del sistema, connettono tutti e tre i bus e forniscono le vie ai dati per la comunicazione con l'esterno. Studiamo ora la funzione dei bus.

I tre bus

Come detto prima, il sistema in Figura 2.1 ha un'architettura a tre bus. Il bus indirizzi ha la funzione di permettere la comunicazione fra la CPU e ROM, RAM o I/O. Il bus dati passa le effettive informazioni fra la CPU e il blocco del sistema abilitato dal bus indirizzi (vedere Figura 2.1). Infine, il bus di controllo svolge due compiti: 1) definisce elettricamente il tipo di comunicazione, 2) inizia e termina il trasferimento.

Nella maggior parte delle comunicazioni la CPU controlla ogni bus. È compito della CPU attivare correttamente ogni bus, in modo da assicurare una comunicazione affidabile. Come programmatori non avete bisogno di preoccuparvi di questa attività, poichè viene svolta dal sistema hardware. Tuttavia si devono fornire le corrette istruzioni alla CPU, in modo che possa generare gli indirizzi e i dati corretti per il sistema hardware (come programmatori si può solo supporre che l'hardware stia funzionando correttamente).

Dimensioni dei bus dell'8086/8088

La larghezza di un bus determina il numero di linee di segnali contenute nell'insieme che compone il bus. Per l'8086/8088 il bus indirizzi può essere largo fino a 20 bit. La larghezza del bus dati è di 16 bit e quella del bus di controllo varia, ma ha un valore nominale di 5 bit. La grandezza di ogni bus differisce da CPU a CPU e da sistema a sistema.

Esaminiamo la Figura 2.2 e discutiamo l'architettura generale interna di un microprocessore. In più mostreremo come l'8086/8088 implementano le loro architetture interne e metteremo in evidenza similitudini e differenze rispetto al caso generale (mostrato in Figura 2.2).

La Figura 2.2 mostra un diagramma di un'architettura standard di un generico microprocessore. Esaminiamo i diversi moduli di questa figura. La scatola di controllo (a destra della figura) rappresenta l'unità di controllo che sincronizza e controlla il sistema hardware.

L'unità aritmetico-logica

Alla sinistra della scatola di controllo c'è l'ALU. L'ALU effettua tutte le operazioni aritmetico-logiche. Alcuni registri speciali, chiamati accumulatori, sono generalmente collegati all'output dell'ALU. Essi contengono i risultati delle operazioni

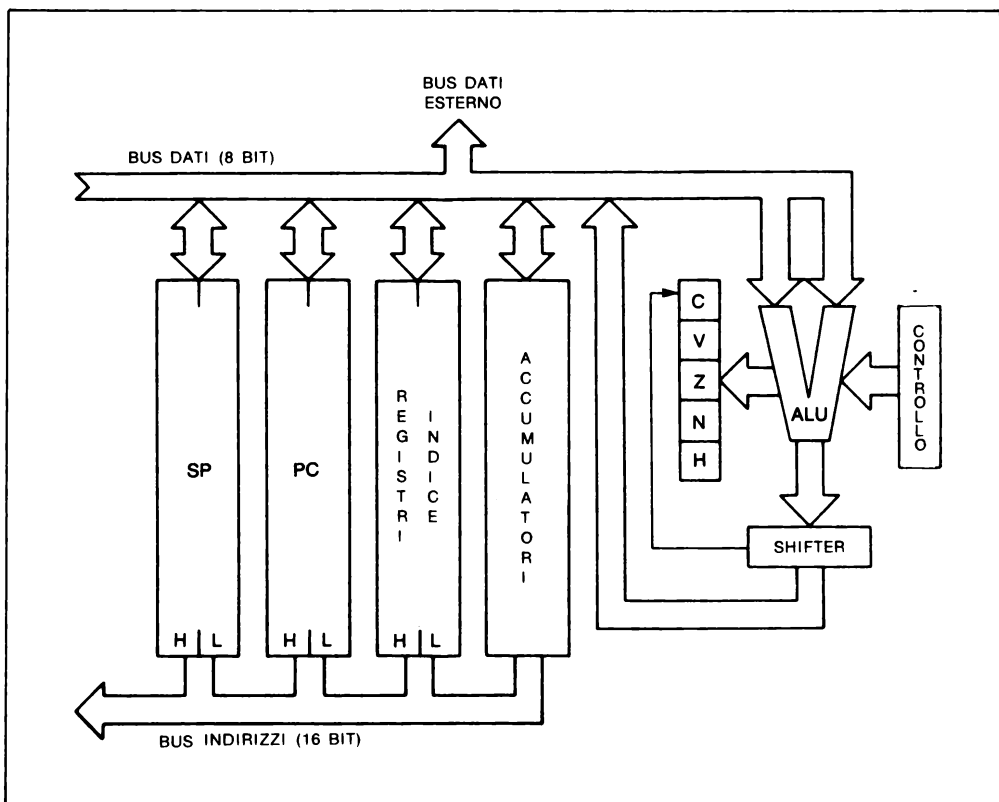



Figura 2.2 — Il diagramma mostra l'architettura interna generale di un microprocessore "standard".

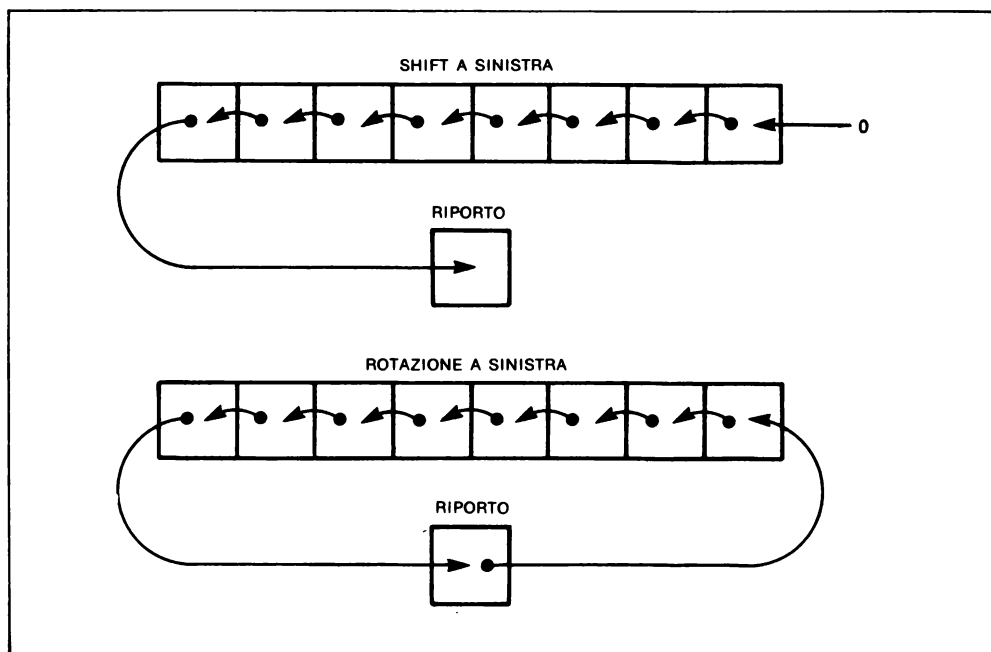



Figura 2.3 — L'ALU fornisce funzioni di shift e di rotazione.

aritmetiche. Oltre alle operazioni aritmetico - logiche, l'ALU fornisce le funzioni di shift e di rotazione. Come illustrato in Figura 2.3, uno shift muove i contenuti dell'accumulatore a sinistra o a destra di uno o più spazi. In questa figura ogni bit è stato spostato a sinistra di una posizione.

Il registro di stato e i flag di condizione

Ritornando alla Figura 2.2, il registro del codice di condizione, o registro di stato, appare alla sinistra dall'ALU. Il compito di questo registro è di immagazzinare le condizioni interne del microprocessore in codici. I codici di condizione sono talvolta chiamati flag di condizione. Un esempio è il bit che indica quando il risultato di un'operazione eseguita dell'ALU lascia tutti i bit dell'accumulatore uguali a zero. Quando ciò accade, il flag di condizione chiamato flag di zero è posto a "vero". I contenuti del registro del codice di condizione possono essere testati con istruzioni speciali. La CPU modifica il suo cammino d'esecuzione basandosi sui valori logici di questi codici. Discuteremo questo argomento in maggior dettaglio quando presenteremo l'insieme di istruzioni e daremo esempi di programmi per l'8086/8088. (NOTA: la maggior parte delle istruzioni eseguite dal microprocessore modificheranno qualcuno o tutti i flag di condizione).

L'insieme di istruzioni per l'8086/8088 (presentate nel Capitolo 4), indicano chiaramente quali istruzioni modificano i flag di condizione e il tipo dei flag modificati.

I registri indirizzi

Spostandoci a sinistra nella Figura 2.2 troviamo i registri indirizzi. Questi registri sono usati per la memorizzazione degli indirizzi e sono collegati al bus indirizzo del sistema. Quando la CPU esegue un programma i registri indirizzi sono combinati logicamente per formare un indirizzo completo del sistema. Parte della potenza del microprocessore deriva dalla sua capacità di combinare i registri indirizzi in modi particolari per formare l'indirizzo di sistema presente nel bus indirizzi. Il capitolo 3 descrive dettagliatamente come i registri indirizzi dell'8086/8088 possono essere combinati logicamente per formare un unico indirizzo. La Figura 2.4 mostra 3 comuni registri indirizzi contenuti in un microprocessore: il contatore di programma (PC), il puntatore allo stack (SP) e il registro indice. Notare che i registri appaiono anche nella Figura 2.2. Analizziamo ora singolarmente questi registri.

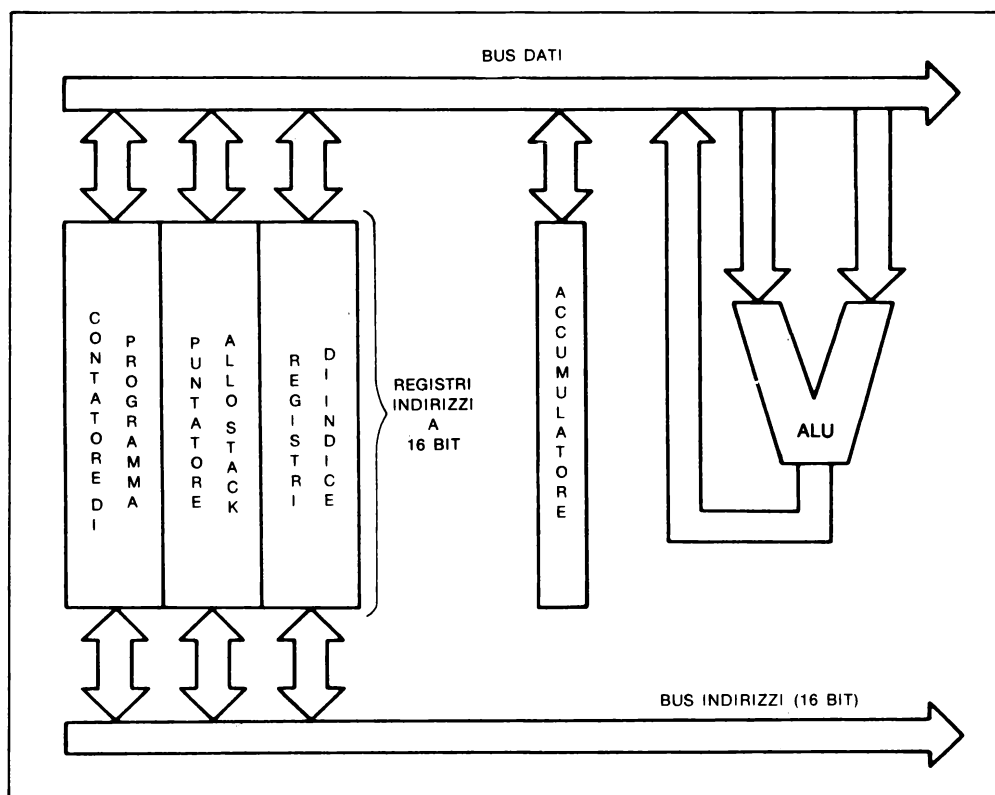



Figura 2.4 — I registri indice PC e SP sono comuni registri di microprocessori usati per formare un indirizzo di sistema completo.

Il contatore di programma (PC)

Il contatore di programma (PC) deve essere presente in tutti i microprocessori, poichè è fondamentale per l'esecuzione del programma. Contiene l'indirizzo della successiva istruzione che deve essere eseguita.

Un programma normalmente si esegue in modo sequenziale. Per accedere all'istruzione successiva è necessario andarla a prendere dalla memoria di sistema e leggerla all'interno del microprocessore. I contenuti del PC sono depositati sul bus indirizzi e mandati alla linea di indirizzo della memoria. La memoria, poi, legge i contenuti specificati dall'indirizzo e manda l'istruzione corrispondente alla CPU.

Il puntatore allo stack (SP)

Il puntatore allo stack (o SP) implementa lo stack del sistema. Lo stack è descritto in dettaglio in una successiva sezione. È usato spesso per trattare le interruzioni e le subroutines e per salvare i dati temporanei). Il puntatore allo stack contiene l'indirizzo di memoria del top dello stack.

Il registro indice

L'indicizzazione è una funzione di indirizzamento della memoria usata per accedere a blocchi di dati nella memoria utilizzando una singola istruzione. Questo metodo non è disponibile in tutti i microprocessori. È tuttavia disponibile nell'8086/8088. Un registro indice contiene tipicamente uno spostamento (displacement) che è addizionato automaticamente a un valore base quando si forma un indirizzo. In breve l'indicizzazione è usata per accedere ad una parola in un blocco di dati. Ora che abbiamo trattato tutti gli elementi della Figura 2.2 proseguiamo con la discussione dello stack.

Lo stack

Lo stack, convenzionalmente definito una struttura LIFO (da Last In, First Out, cioè l'ultimo elemento inserito sarà il primo ad essere prelevato), è un insieme di locazioni di memoria allocate in una struttura di dati a pila. La caratteristica essenziale dello stack è che il primo elemento introdotto è sempre in fondo e che l'ultimo elemento inserito è sempre in alto.

Un'analogia può essere fatta con i piatti impilati su di un

banco di un ristorante dove si suppone ci sia un buco con una molla sul fondo e che i piatti siano inpilati in questo buco. Con questa organizzazione si garantisce che il piatto posto nella catasta per primo è sempre in fondo e che quello messo per ultimo è sempre in cima. Nell'uso normale uno stack è accessibile solo per mezzo di due operazioni o istruzioni: PUSH e POP.

Operazioni di accesso allo stack

Queste due istruzioni sono illustrate nella Figura 2.5. L'operazione PUSH deposita un elemento in cima allo stack. L'operazione POP trasferisce l'elemento in cima allo stack nel registro interno specificato dall' istruzione.

IL CICLO GENERALE DI UN'ISTRUZIONE

Esaminiamo la Figura 2.6; questo diagramma illustra il ruolo del contatore di programma che preleva un'istruzione dalla memoria. L'unità microprocessore appare sulla sinistra dell'illustrazione e la memoria sulla destra. La memoria immagazzina istruzioni e dati. Il chip di memoria può essere sia ROM che RAM, o qualsiasi altro chip che contiene memoria.

Guardando la Figura 2.6 supporremo che il contatore di programma contenga un indirizzo che è l'indirizzo della prossima istruzione che deve essere prelevata dalla memoria.

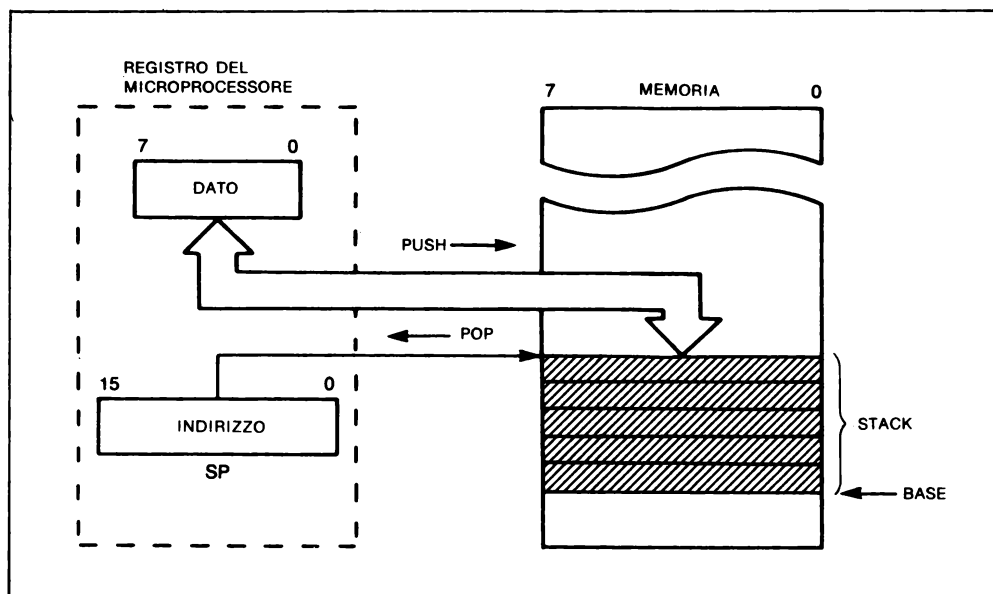



Figura 2.5 — Il diagramma mostra le due istruzioni di manipolazione dello stack: PUSH e POP.

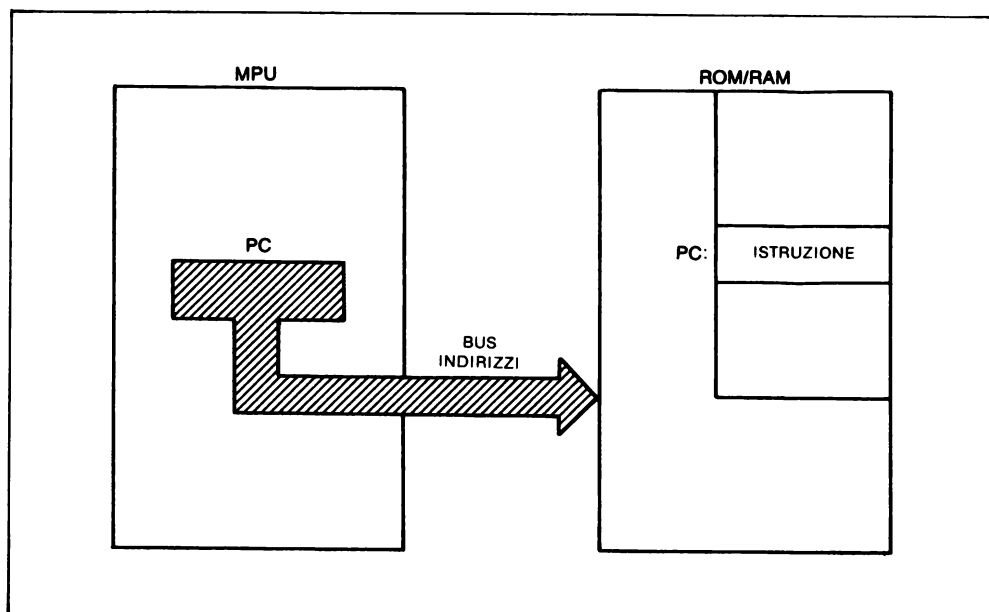



Figura 2.6 — Il diagramma mostra il reperimento di un'istruzione dalla memoria.

Ogni microprocessore opera in tre cicli che comprendono:

1) il reperimento dell'istruzione successiva;
2) la decodifica dell'istruzione;
3) l'esecuzione dell'istruzione

Analizziamo in dettaglio ogni ciclo.

Reperimento dell'istruzione (fetching)

Nel primo ciclo i contenuti del contatore di programma sono posti nel bus indirizzi e mandati alla memoria al momento giusto. Il bus di controllo genera allora un segnale di lettura della memoria. Quando la memoria riceve il segnale di lettura, il dato contenuto in memoria all'indirizzo specificato è posto nel bus dati del sistema. Il microprocessore allora legge l'informazione dal bus dati e la scrive in un registro interno chiamato registro istruzione (IR).

L'informazione letta nella CPU è l'istruzione. Possiamo dire che l'istruzione è stata reperita dalla memoria. La Figura 2.7 è un diagramma a blocchi che mostra l'operazione di reperimento di un'istruzione dalla memoria.

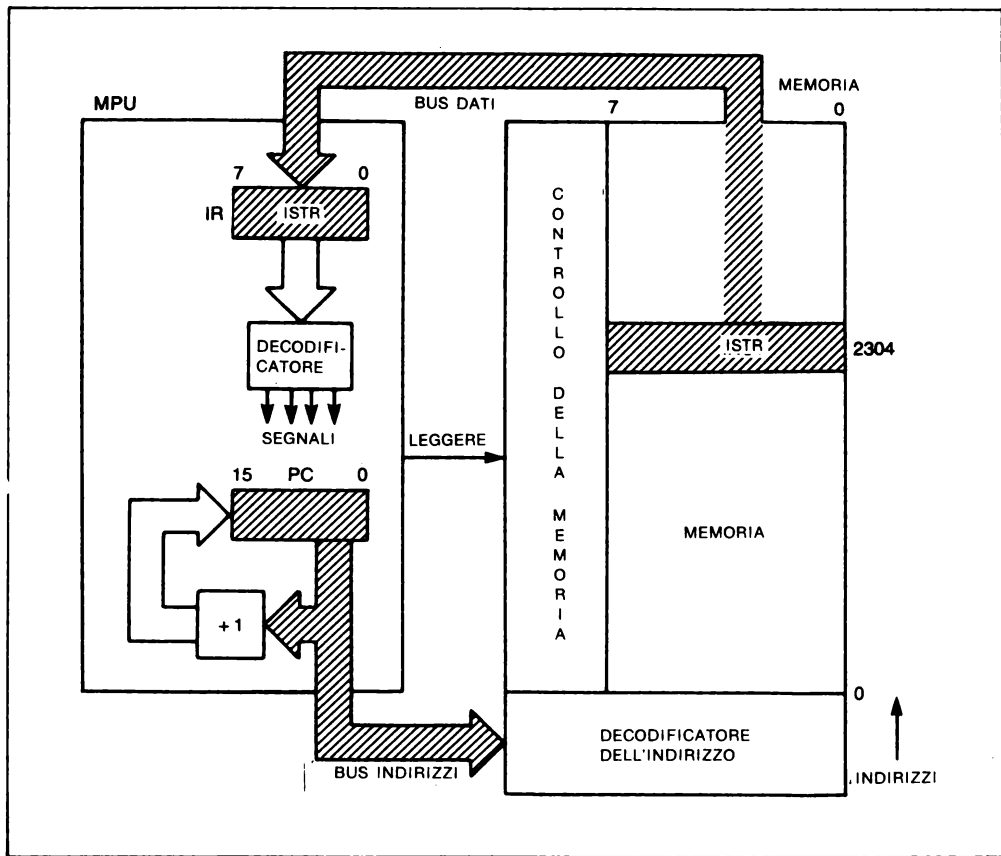



Figura 2.7 — Il diagramma mostra il reperimento di un'istruzione dalla memoria.

Decodifica ed esecuzione

Una volta che l'istruzione è nell'IR, l'unita' di controllo del microprocessore la decodifica e genera una corretta sequenza di segnali interni ed esterni per eseguirla. La CPU impiega un certo tempo, normalmente un periodo di clock, per decodificare un'istruzione e decidere logicamente che azione compiere. Potremo vedere più avanti che l'8086/8088 utilizza bene questo tempo morto (cioè il tempo aspettato perchè la CPU decodifichi l'istruzione).

Misura della velocità di esecuzione di un'istruzione

Una volta che la CPU ha decodificato l'istruzione compie una serie di azioni dettate dall'istruzione. Alcune istruzioni richiedono compiti semplici, così ci vuole poco tempo per eseguirle; altre richiedono che avvengano diverse azioni, per cui richiedono molto tempo. La velocità di esecuzione per un'i-

struzione è generalmente espressa dal numero totale di cicli di clock richiesti per il suo completamento.

Reperimento dell'istruzione successiva

Abbiamo appena descritto come viene reperita un'istruzione dalla memoria usando il contatore di programma. Durante l'esecuzione di un programma le istruzioni sono prelevate in sequenza. È perciò necessario un meccanismo automatico, chiamato incrementatore, per prelevare le istruzioni in sequenza. L'incrementatore è collegato al contatore di programma come mostra la Figura 2.7.

Deve essere sottolineato che le precedenti descrizioni sono semplificate. Le istruzioni possono essere lunghe 2, 3 o 6 byte e in tali casi il PC deve decidere logicamente quanti byte sono contenuti in ogni istruzione prima che cominci l'istruzione successiva. Discuteremo ancora sulla funzione del PC quando esamineremo in dettaglio l'architettura interna dell'8086/8088.

ORGANIZZAZIONE INTERNA DELL'8086/8088

Finora abbiamo discusso (in termini generali) come è organizzato il sistema di un microprocessore e abbiamo esaminato la struttura interna di una CPU. Ora, basandoci su queste informazioni, descriviamo in dettaglio la struttura interna dell'8086/8088. Una volta capito questo soggetto è molto più facile comprendere l'argomento dei modi di indirizzamento (affrontato nel capitolo successivo).

Le architetture interne dell'8086/8088 sono virtualmente identiche dal punto di vista della programmazione. Il software scritto per l'8086 può essere eseguito dall'8088 senza nessuna modifica. Ed e' vero anche il contrario. (NOTA: è importante notare anche che l'implementazione hardware di ciascun microprocessore è abbastanza unica ma questo ci riguarda poco poichè in questo testo ci stiamo concentrando sulla programmazione.)

La Figura 2.8 mostra un diagramma a blocchi funzionale dell'8086/8088. Questo diagramma mostra che ci sono due principali blocchi funzionali logici: la BIU (unità di interfaccia con i bus) e l'EU (unità di esecuzione). Nel diagramma a blocchi generale della Figura 2.2 entrambe le funzioni erano raggruppate in un singolo blocco di controllo.

L'unità di interfaccia con i bus (BIU)

Esaminiamo le loro funzioni. La principale della BIU nell'8086/8088 è di fornire l'interfaccia fisica tra l'8086/8088 e il mondo "esterno". La BIU controlla gli indirizzi del sistema,

il bus dati e il bus di controllo. Può operare in parallelo con l'EU. Una delle caratteristiche uniche della BIU è la sua capacità di "reperire in anticipo le istruzioni". Per capire ciò che significa consideriamo il seguente fatto. Quando la CPU esegue le istruzioni, prima va a prendere dalla memoria l'istruzione e poi la decodifica. Durante il periodo di decodifica i bus esterni della CPU sono inattivi; in altre parole non c'è alcuna attività sui bus poichè la CPU non sa "elettricamente" cosa fare dato che l'istruzione non è stata ancora decodificata.

Reperimento anticipato dell'istruzione successiva

Nell'8086/8088, la BIU va a prendere un'istruzione dalla memoria e poi, quando l'EU sta ancora decodificando l'istruzione ricevuta, la BIU va a prendere l'istruzione successiva dalla memoria. In questo modo l'istruzione successiva è prelevata dalla memoria in anticipo e attende di essere eseguita dalla CPU. In pratica, l'8086 può avere fino a 6 byte di informazione

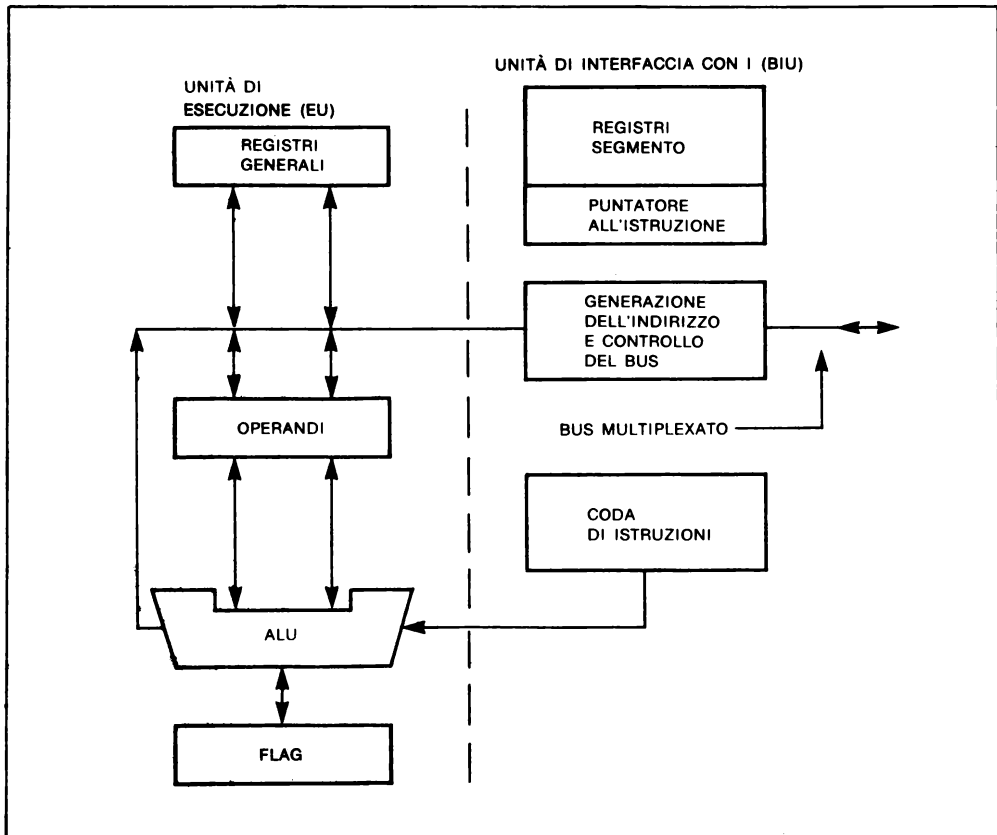



Figura 2.8 — Il diagramma mostra i due blocchi logici funzionali di base all'interno dell'8086/8088. Essi sono l'EU (unità di esecuzione) ed il BIU (unità di interfaccia con i bus).

nella coda delle istruzioni, mentre l'8088 può averne fino a 4. Alcune istruzioni dell'8086/8088 sono eseguite completamente all'interno della CPU, e possono richiedere molti cicli di clock. Un esempio di questo tipo di istruzioni è l'istruzione DIVIDE. Durante il tempo in cui la CPU esegue la divisione, la BIU preleva dati dalla memoria fino al numero massimo di byte che può essere contenuto nella coda delle istruzioni. Prendendo i dati dalla memoria in anticipo si riduce il tempo globale di esecuzione delle istruzioni. Questo perchè il tempo per andare a prendere le istruzioni non deve essere conteggiato nel tempo globale. Questa affermazione tuttavia non è sempre vera poichè ci saranno volte in cui un'intera istruzione deve essere ottenuta senza poter essere presa in anticipo. Questo accade ogni volta che la coda è ripulita o ricomposta. Per esempio, ogni volta che l'8086/8088 esegue un'istruzione di salto la coda viene ripulita e deve essere quindi ricomposta di nuovo. Se non avete familiarità con l'istruzione di salto, potreste non capire perchè la coda debba essere ripulita. Questo sarà chiarito studiando l'insieme di istruzioni nel Capitolo 4.

L'unità di esecuzione (EU)

L'altra sezione funzionale hardware dell'8086/8088 mostrata in Figura 2.8 è un'unità di esecuzione. Questa unità contiene un'unità logico-aritmetica a 16 bit. Mantiene i flag di condizione della CPU. L'EU maneggia inoltre i registri generali della CPU. Tutti i registri dell'EU sono larghi 16 bit. Questo fatto è vero sia per l'8086 che per l'8088.

I REGISTRI GENERALI DELL'8086/8088

La Figura 2.9 mostra i registri generali dell'8086/8088.

I Registri Dati

Come possiamo vedere, in questa figura ci sono due gruppi principali: il gruppo dati, e il gruppo puntatori e indici. Ogni registro è largo 16 bit. I registri del gruppo dati sono etichettati con AX, BX, CX, e DX. Ognuno di essi può essere usato sia come singolo registro a 16 bit che come 2 registri di 8 bit ciascuno. Quando vengono usati come 2 registri ad 8 bit sono divisi in una metà superiore (H) e in una inferiore (L), e vengono chiamati AH, AL, BH, BL, CH, CL, DH, DL.

I Registri Puntatori e Indici

I registri del gruppo puntatori e indici (mostrati in Figura 2.9) sono usati solo come registri a 16 bit. Questi registri sono etichettati SP (puntatore allo stack) BP (puntatore di base) SI (indice sorgente) e DI (indice destinazione). Il ragionamento dietro questi nomi diventerà chiaro più avanti nel testo quando discuteremo i vari modi di indirizzamento e mostreremo esempi di programmi per l'8086/8088. Per ora è solo importante sapere che questi registri esistono e come sono organizzati.

I registri di segmentazione

Ci sono 4 registri di segmentazione della memoria nell'8086/8088, come mostrato in Figura 2.10 . Daremo ora una breve descrizione dei registri di segmentazione, rimandando al Capitolo 3 una spiegazione più dettagliata.

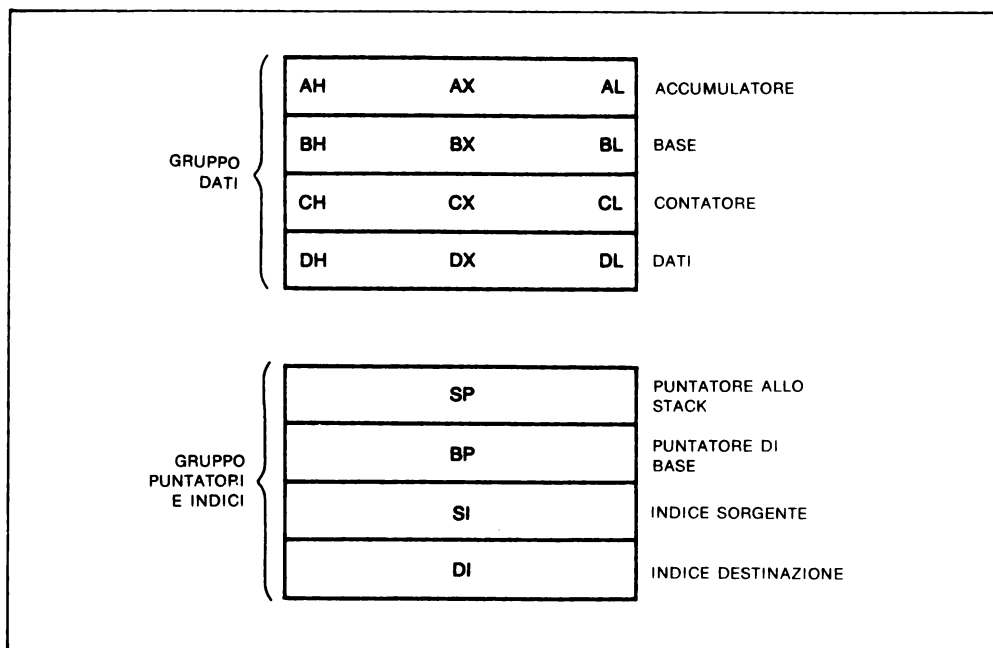



Figura 2.9 — Il diagramma mostra i registri interni generali dell'8086/8088.

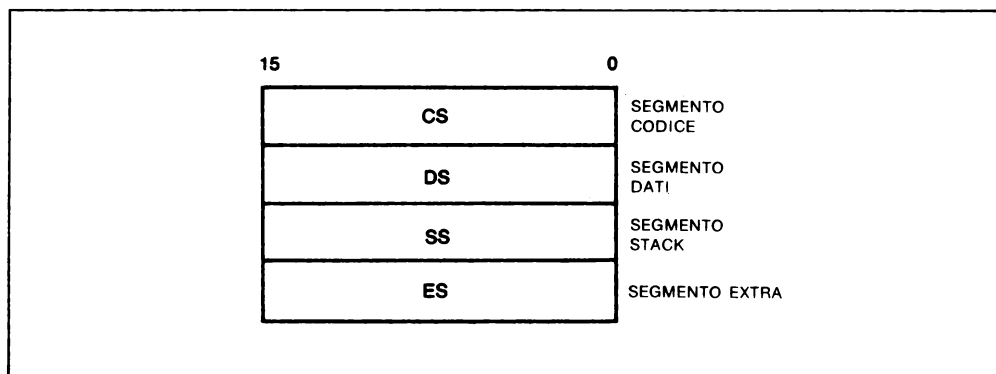



Figura 2.10 — Questo diagramma mostra i quattro registri di segmentazione dell'8086/8088. Questi registri sono combinati logicamente con i registri generali per formare un indirizzo del sistema a 20 bit.

Ogni registro di segmentazione è lungo 16 bit ed ha uun nome particolare: Segmento Codice (CS), Segmento Dati (DS), Segmento Stack (SS), Segmento Extra (ES). I nomi sono utili per capire come i registri di segmentazione siano usati durante l'esecuzione di un programma.

Funzione dei Registri di Segmentazione

Questi registri sono combinati con altri registri interni durante l'esecuzione del programma per formare un indirizzo di sistema completo di 20 bit per il bus indirizzi esterno della CPU. Per esempio quando si effettua un'operazione "push" o "pop" con lo stack il registro SS è combinato logicamente con altri registri per formare l'indirizzo di memoria del top dello stack. Ci dilungheremo su questo nella discussione dei modi di indirizzamento nel Capitolo 3.

Il puntatore all'istruzione (IP)

Nella descrizione del diagramma a blocchi generale di un microprocessore mostrato nella Figura 2.2 abbiamo discusso la funzione del contatore di programma. Nell'8086/8088 il contatore di programma è sostituito dal registro puntatore all'istruzione (IP). Il registro IP è aggiornato dalla BIU in modo tale da indicare l'indirizzo dell'istruzione successiva. I programmi non possono accedere direttamente all'istruzione; ma durante l'esecuzione di un programma, l'IP può essere modificato o salvato nello stack e recuperato successivamente.

I flag

Gli 8086/8088 hanno 6 flag di condizione, o flag di stato, a 1 bit che sono aggiornati dall'EU. Questi flag, mostrati in Figura 2.11, rappresentano la condizione del risultato di un'operazione aritmetica o logica che è appena avvenuta.

Nel Capitolo 4 troverete un gruppo di istruzioni che permetteranno al programma di alterare la sua sequenza di esecuzione in base al valore logico di questi flag.

Definizione dei flag

AF: *flag di riporto ausiliario.* Se questo flag è postoa uno c'è stato un riporto del nibble inferiore a quello superiore o un prestito dal nibble superiore a quello inferiore. Il nibble superiore e quello inferiore si riferiscono al byte di ordine inferiore di un valore a 16 bit.

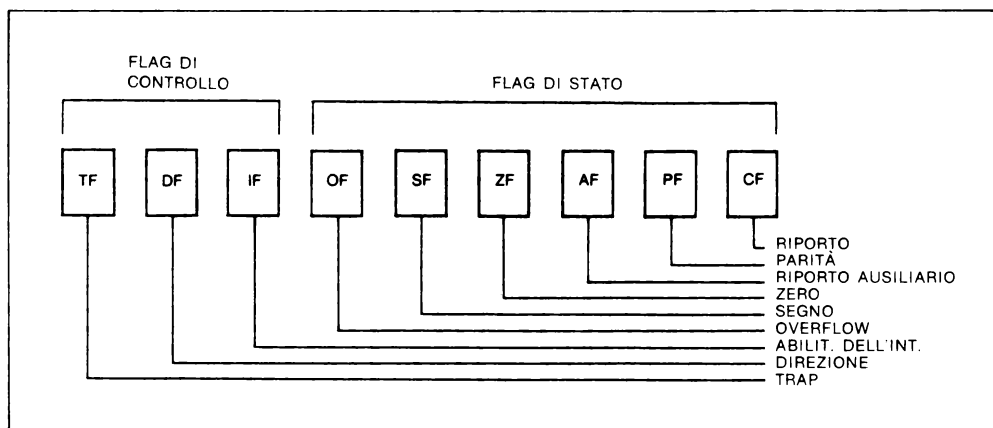



Figura 2.11 — Flag del sistema 8086/8088.

CF: *flag di riporto.* Questo flag è posto a uno quando c'è stato un riporto o un prestito dal bit di ordine alto del risultato (a 8 o 16 bit). (Abbiamo discusso il flag generale di riporto nel Capitolo 1).

OF: *flag di overflow.* Quando questo vale uno si è verificato un overflow aritmetico. Questo significa che la dimensione del risultato supera la capacità della cella di destinazione e che una cifra significativa è stata persa. (Il flag generale di overflow è stato discusso nel Capitolo 1).

SF: *flag di segno.* Questo flag è posto a uno quando il bit d'ordine superiore del risultato è un 1 logico. Poichè i numeri binari negativi sono rappresentati usando la notazione in complemento a 2, SF riflette il segno del risultato: 0 indica un numero positivo e 1 un numero negativo.

PF: *flag di parità.* Se questo flag ha il valore uno significa che il risultato dell'operazione contiene un numero pari di 1. Questo flag può essere usato per controllare gli errori nella trasmissione dei dati.

ZF: *flag di zero.* Questo flag viene posto a uno quando il risultato di un'operazione è zero.

La Figura 2.12 mostra l'insieme completo dei registri interni dell'8086/8088.

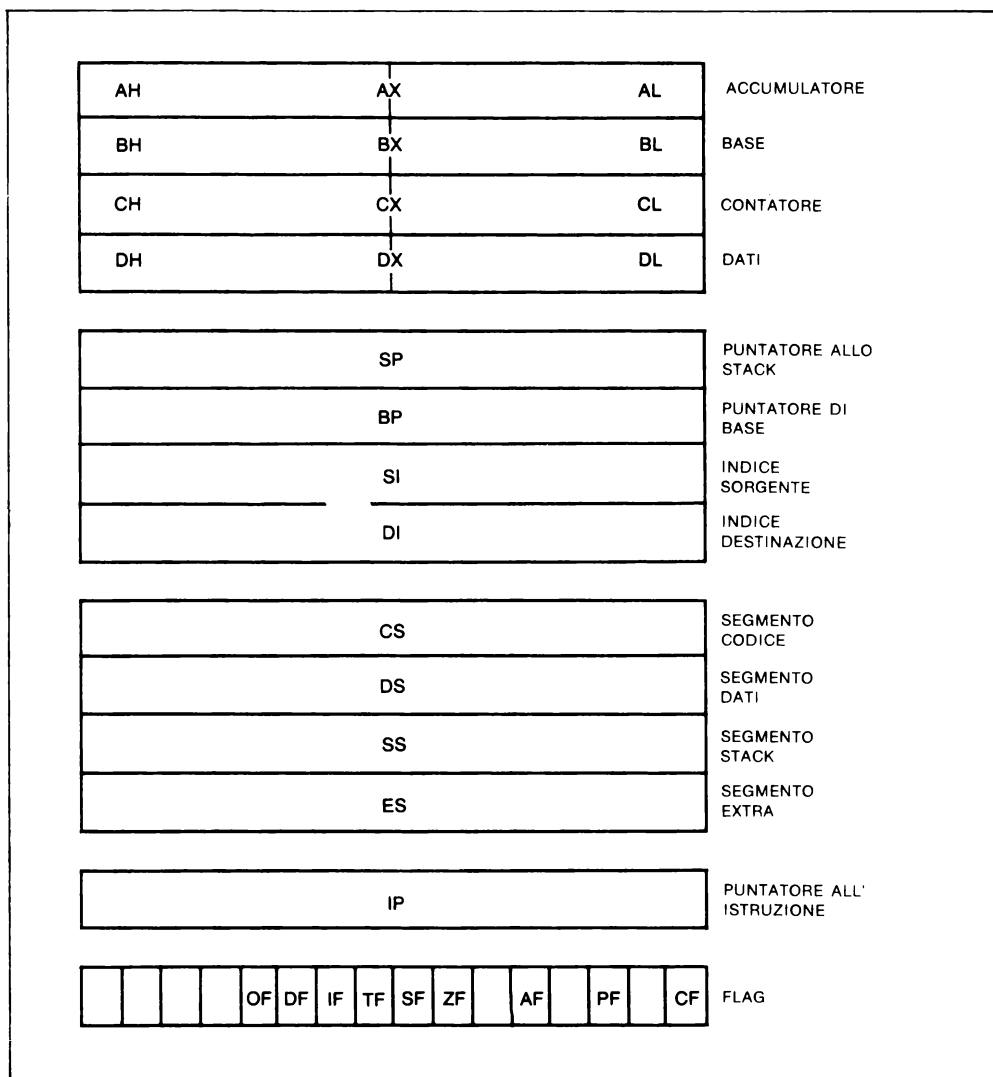



Figura 2.12 — Il diagramma mostra i registri interni dell'8086/8088.

FLAG ADDIZIONALI DI CONTROLLO

Ci sono tre flag di controllo nell'8086/8088 che non abbiamo ancora trattato. Abbiamo scelto di non menzionarli in questo momento perchè sarà più utile trattarli quando ne sapremo di più sull'insieme di istruzioni dell'8086/8088. Questi tre flag sono chiamati DF (flag di direzione), IF (flag di abilitazione delle interruzioni), TF (flag di trap).

SOMMARIO

In questo capitolo abbiamo trattato l'organizzazione base del sistema microprocessore. Abbiamo descritto ROM, RAM, I/O e abbiamo spiegato come l'architettura a tre bus è interfaccia ad essi. Abbiamo definito ogni bus secondo le sue funzioni nel sistema. Inoltre abbiamo esaminato il diagramma a blocchi interno di un microprocessore. Abbiamo esplorato ogni blocco principale e le sue interazioni con gli altri blocchi. Infine abbiamo discusso l'organizzazione interna dell'8086/8088 cominciando dai blocchi principali: la CPU, la BIU e l'EU; abbiamo poi rivolto la nostra attenzione ai registri interni dell'8086/8088. Nel prossimo capitolo useremo queste informazioni per approfondire la nostra conoscenza dell'8086/8088. In particolare esamineremo come l'8086/8088 usa i registri interni per formare uno specifico indirizzo di memoria.

CAPITOLO 3

ORGANIZZAZIONE DELLA MEMORIA DELL'8086/8088 E MODI DI INDIRIZZAMENTO

INTRODUZIONE

In questo capitolo esamineremo l'organizzazione della memoria di sistema dei microprocessori 8086/8088.

Sottolineeremo le molte simiglianze e diversità fra le due architetture e discuteremo i differenti modi di indirizzamento di ognuno. Inoltre daremo degli esempi che mostrano come ciascun modo di indirizzamento opera. Infine impareremo come generare un codice oggetto o binario per l'8086/8088 e discuteremo vari punti importanti che riguardano la scrittura del codice oggetto.

ORGANIZZAZIONE DELLA MEMORIA DELL'8086

Larghezza della memoria: byte superiore e byte inferiore

L'8086 ha 20 linee di indirizzo. Questo significa che ha 2^{20} (cioè 1048576) singole locazioni di memoria che possono essere indirizzate. Poichè il bus dati per l'8086 è largo 16 bit, la memoria di sistema per l'8086 è larga 16 bit (o 2 byte). Questi 2 byte sono chiamati byte superiore e byte inferiore, come mostrato in Figura 3.1.

L'8086 può accedere a ciascuno dei possibili 1048576 byte di memoria. Poichè ogni parola di memoria usa 2 byte, si ha una memoria del sistema di 2^{19} indirizzi di 16 bit ciascuno (NOTA: è importante ricordare che l'8086 può accedere sia ai byte di memoria che alle parole.

Come vedremo nel capitolo seguente, le istruzioni per l'8086 sono composte di dati da 1 a 6 byte. Questo significa che le istruzioni dell'8086 possono incominciare o da un indirizzo pari (byte inferiore) o da uno dispari (byte superiore), come

BUS DATI A 16-BIT

BYTE SUPERIORE

BYTE INFERIORE

MEMORIA DEL SISTEMA

BUS INDIRIZZI A 20 BIT

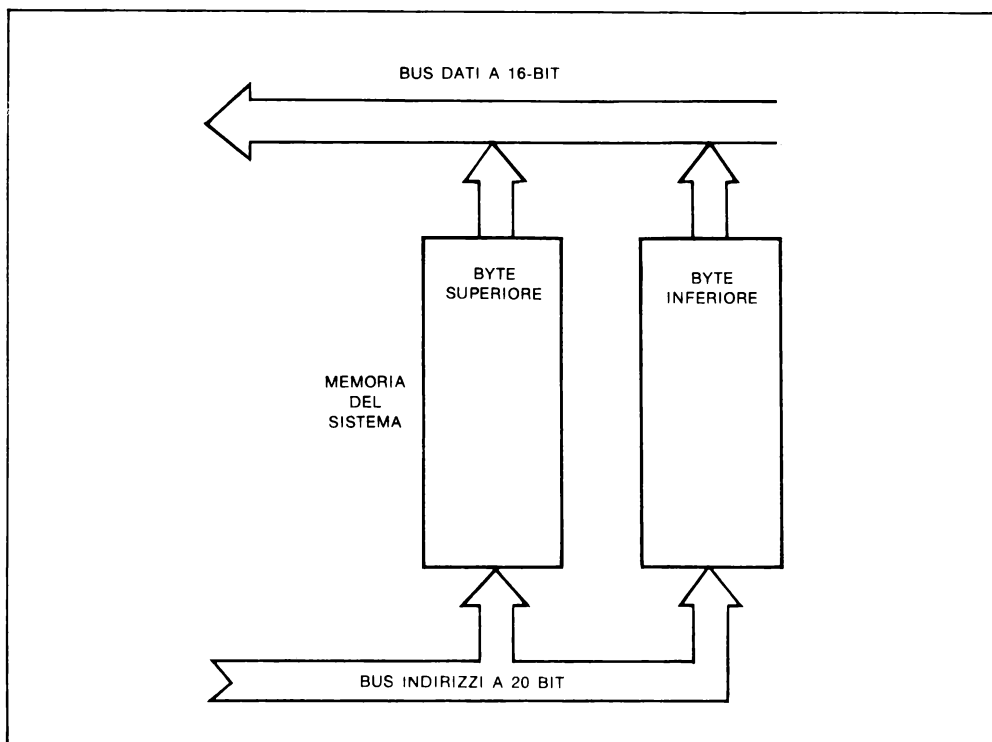



Figura 3.1 — Il diagramma mostra come i 16 bit della memoria dell'8086 vengono divisi in byte superiore ed in byte inferiore.

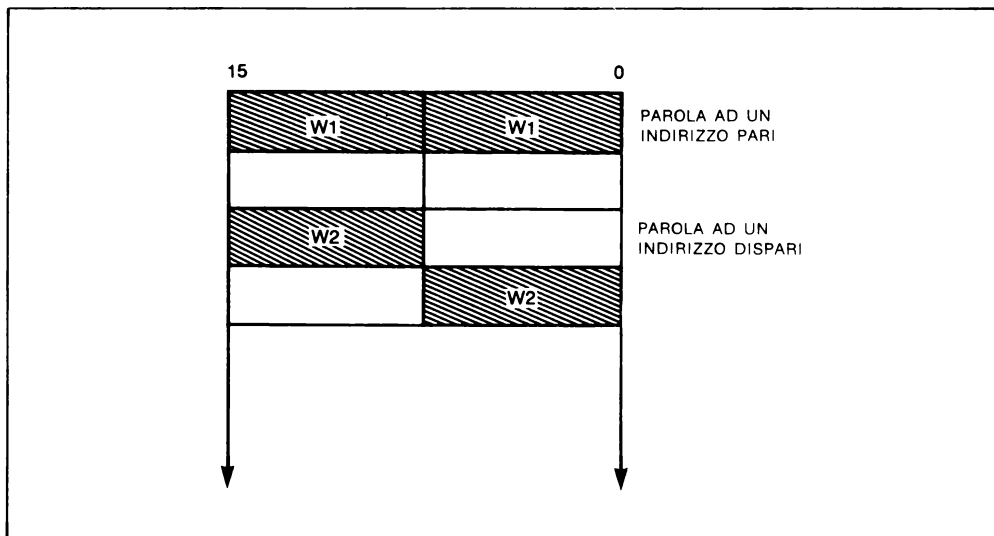



Figura 3.2 — Le parole (16 bit) immagazzinate nella memoria dell'8086 possono risiedere sia in un indirizzo pari che in uno dispari.

Determinazione del byte a cui accedere

mostrato nella Figura 3.2. I due segnali A0 e BHE determinano a quale byte di memoria si accede.

Quando A0 è uno zero logico, l'8086 sta accedendo al byte inferiore dell'indirizzo di memoria valido. Quando BHE è uno zero logico, l'8086 sta accedendo al byte superiore. Da ciò si può essere portati a pensare che BHE sia semplicemente $\overline{A0}$. Questo può essere vero in alcuni casi, ma non in tutti.

L'8086 è capace di accedere a 16 bit (o due byte) di dati nella memoria in un unico ciclo di memoria. Quando ciò accade sia BHE che A0 sono degli zero logici nello stesso istante. Come programmatori non si è tenuti a conoscere questi segnali; tuttavia è importante che si sappia che qualsiasi byte della memoria può essere accessibile separatamente quando sono usate alcune istruzioni dell'8086.

L'accesso a 16 bit ad un indirizzo pari

Per spiegare più dettagliatamente come l'8086 accede a dati nella memoria esaminiamo alcuni esempi. Supponiamo che l'8086 stia leggendo una parola di 16 bit dall'indirizzo 000F0, un indirizzo pari. Questo significa che l'8086 richiede sia il byte inferiore che quello superiore dei dati in quell'indirizzo. La CPU mette l'indirizzo nel bus indirizzi con BHE e A0 uguali a uno zero logico. Entrambi i byte comunicano con la CPU allo stesso tempo (in modo parallelo come risulta dalla Figura 3.3).

L'accesso ad 8 bit ad un indirizzo pari

Supponiamo che l'8086 stia leggendo un byte dall'indirizzo di memoria 000F0. Poichè questo indirizzo è pari, il byte di dati sarà letto dal byte inferiore dell'indirizzo della parola di dati. In questo caso A0 è uguale a uno zero logico e BHE è uguale ad un 1 logico.

L'accesso ad 8 bit ad un indirizzo dispari

Supponiamo che l'8086 stia accedendo a un byte all'indirizzo 000F1, un indirizzo dispari. Questo dato sarà letto dal byte superiore della parola dati di 16 bit all'indirizzo 000F0. In questo esempio, A0 è un 1 logico e BHE uno 0 logico.

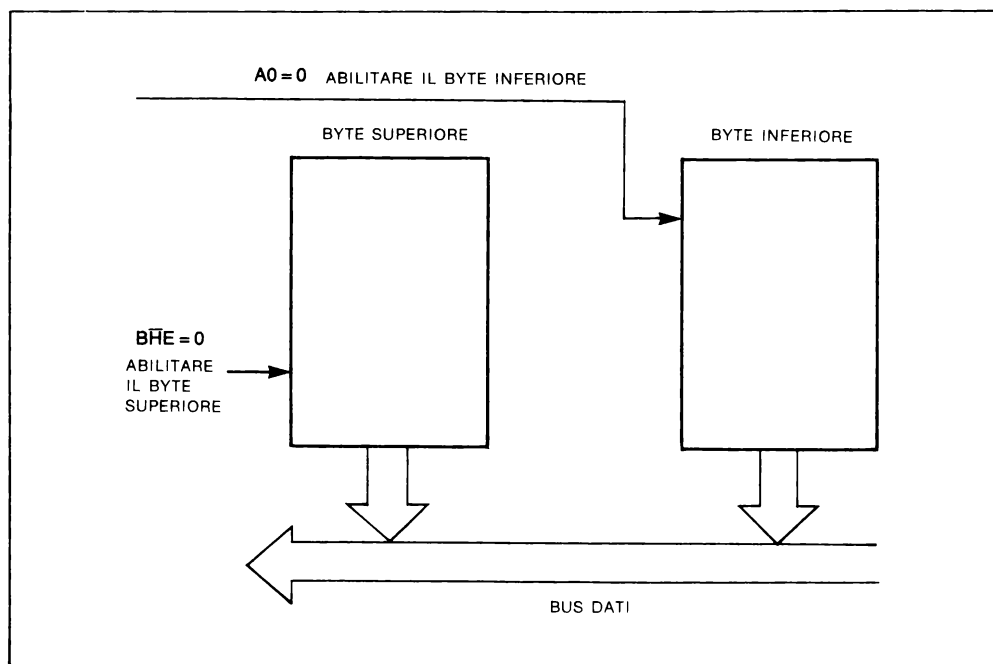



Figura 3.3 — Quando l'8086 accede ad una parola di 16 bit che risiede ad un indirizzo pari della memoria del sistema entrambi i byte superiore ed inferiore vengono abilitati nel bus dati del sistema; in questo caso A0=0 e BHE=0.

L'accesso a 16 bit ad un indirizzo dispari

Finora abbiamo descritto due casi in cui l'8086 sta accedendo a dati nella memoria e le condizioni logiche di A0 e BHE sono diverse. Ci si deve porre la seguente domanda: ".PA Che cosa accade se l'8086 accede a una parola di 16 bit che comincia ad un indirizzo dispari?". Questo problema è mostrato in Figura 3.4.

Questo è un tipico problema dovuto alla flessibilità della struttura di memoria dell'8086. Le istruzioni possono richiedere un numero pari o dispari di byte. Per evitare uno spreco di byte, l'8086 permette che i byte o le parole di dati possano cominciare ad un indirizzo pari o ad uno dispari. Così, per leggere una parola di 16 bit che comincia con un indirizzo dispari l'8086 deve effettuare due accessi alla memoria.

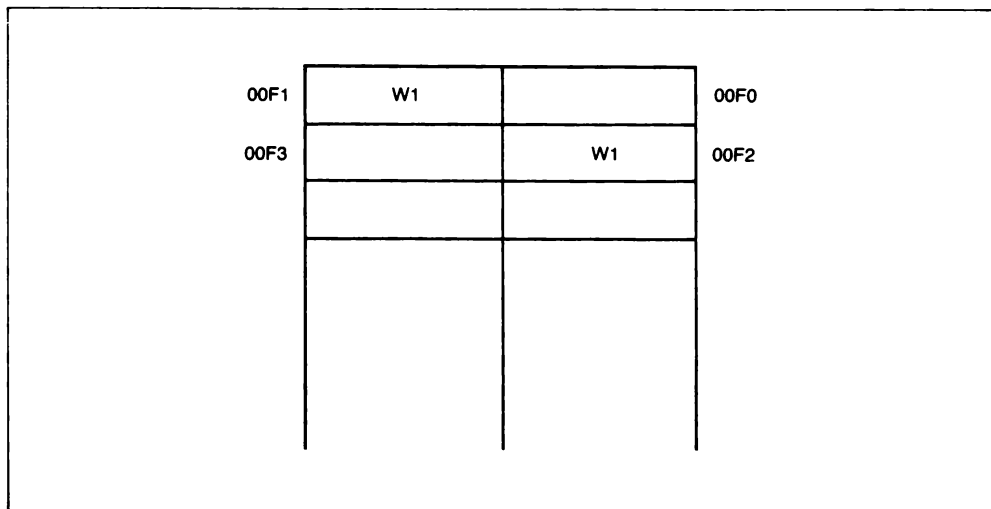



Figura 3.4 — Se una parola è immagazzinata ad un indirizzo dispari, l'8086 accede l'intera parola compiendo due accessi alla memoria.

PAROLE ALLINEATE E NON ALLINEATE

Ora conosciamo i diversi modi in cui l'8086 accede a dati nella memoria. Quando la CPU accede a una parola (cioe' a 16 bit) posta ad un indirizzo pari sta accedendo a una parola "allineata". La parola è allineata perchè entrambi i byte si trovano allo stesso indirizzo di parola e possono essere letti o scritti in un unico ciclo di memoria.

Quando la CPU accede a una parola che comincia ad un indirizzo dispari si dice che accede ad una parola non allineata. Questo è dovuto al fatto che i due byte della parola non si trovano nello stesso indirizzo di parola. Così sono richiesti due cicli di memoria per leggere l'intera parola. La Figura 3.5 mostra il concetto di parola allineata e non allineata.

Influenza dell'allineamento sulla velocità di esecuzione delle istruzioni

L'8086 è in grado di gestire sia le parole allineate che quelle non allineate. L'importanza che le parole siano o no allineate è dovuta al fatto che l'allineamento e il non allineamento determinano la velocità di esecuzione di certe operazioni. Per esempio se un programma accede o utilizza una notevole quantità di parole, allora memorizzando i dati facendoli cominciare ad indirizzi pari si può avere una velocità di esecuzione maggiore rispetto al caso in cui le parole sono non allineate.

Figura 3.5 — Il diagramma mostra il concetto di parola allineata e non allineata. Si può accedere a parole allineate in un singolo ciclo di accesso alla memoria.

ORGANIZZAZIONE DELLA MEMORIA DELL'8088

Come la CPU 8086, anche la CPU 8088 ha 20 bit di indirizzo. Diversamente, l'8088 ha un bus dati di soli 8 bit. Questo porta ad un'organizzazione della memoria simile a quella mostrata nella Figura 3.6. L'8088 è essenzialmente un sistema a 8 bit che usa un microprocessore a 16 bit. Comunque, tutte le possibilità dal software a 16 bit sono disponibili anche nell'8088. Questo è dimostrato dal fatto che sia l'8086 che l'8088 eseguono le stesse istruzioni software.

GENERAZIONE DI UN INDIRIZZO CON L'8086 E L'8088

Nelle precedenti discussioni abbiamo introdotto l'organizzazione della memoria sia per la CPU 8086 che per la CPU 8088.

Nel Capitolo 2 abbiamo discusso i registri interni per questi microprocessori. Nella rimanente parte di questo capitolo impareremo come sono generati gli indirizzi usando i registri in-

Cos'è un modo di indirizzamento

terni della CPU. Ogni diverso metodo per generare un indirizzo è detto modo di indirizzamento.

Generazione dell'indirizzo di un'istruzione

Vediamo dapprima come l'8086/8088 genera un indirizzo per un'istruzione nella memoria. Ricordate dalle precedenti discussioni che tutti i registri interni dell'8086/8088 sono larghi 16 bit. Tuttavia il bus fisico di indirizzo della CPU è largo

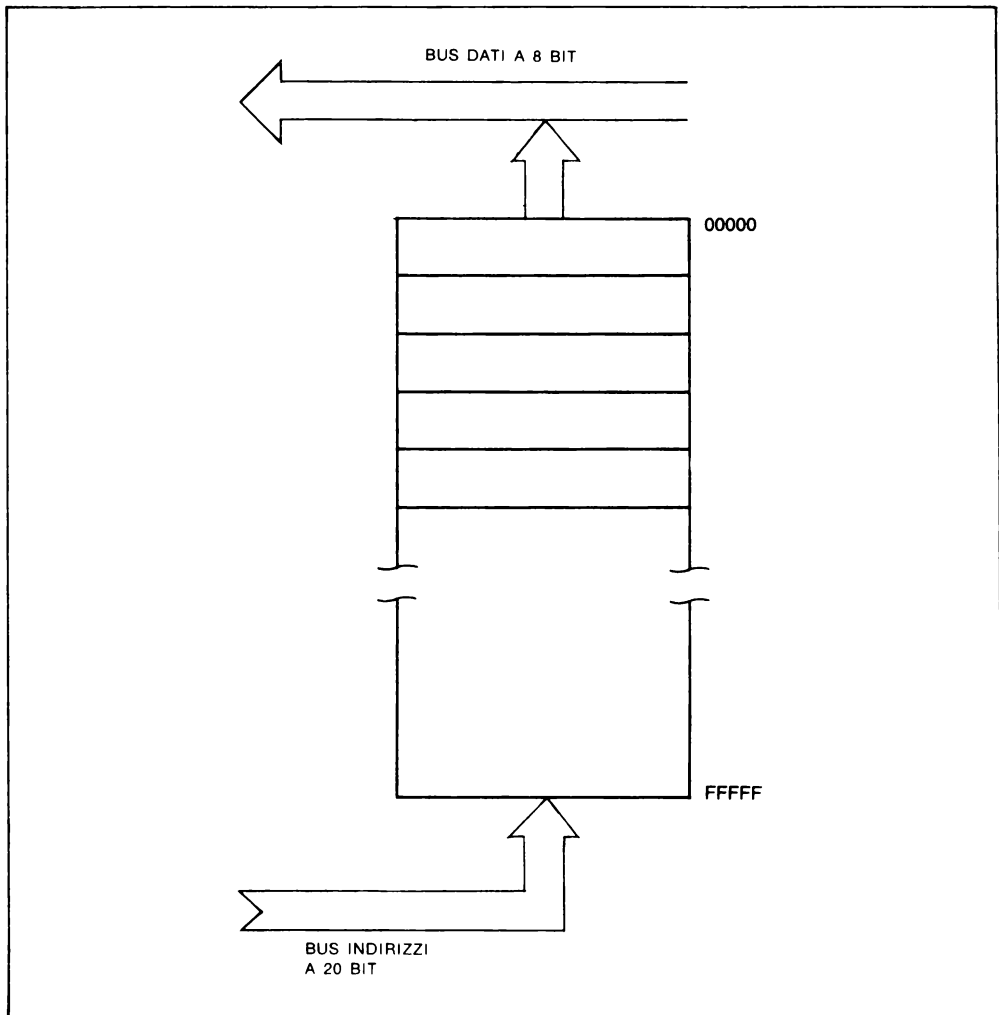



Figura 3.6 — Questo diagramma rappresenta la memoria del sistema 8088.

20 bit. Questo significa che deve essere usato più di uno dei registri interni della CPU per generare un indirizzo fisico di 20 bit. I 2 registri usati per l'indirizzo di un'istruzione sono l'IP (registro del puntatore all'istruzione) e il CS (registro del segmento codice). Questi 2 registri sono combinati in un modo speciale per generare l'indirizzo completo di 20 bit. Ecco l'equazione che descrive come questi 2 registri di 16 bits sono combinati: indirizzo di 20 bit (16 x CS) + IP

Generazione di un indirizzo fisico con i registri IP e CS

Per esempio: registro CS = 1000H
registro IP = 0414H

perciò indirizzo di 20 bit = (16 x 1000H) =

= 10000H [spostato (shift) a sinistra di 4 bit] + 0414H =
= 10414H
(La H indica che gli indirizzi sono espressi in esadecimale)

Valore di offset e indirizzo di base

Questo è l'indirizzo nella memoria dal quale la nuova istruzione sarà presa. Il registro IP è chiamato offset — il registro

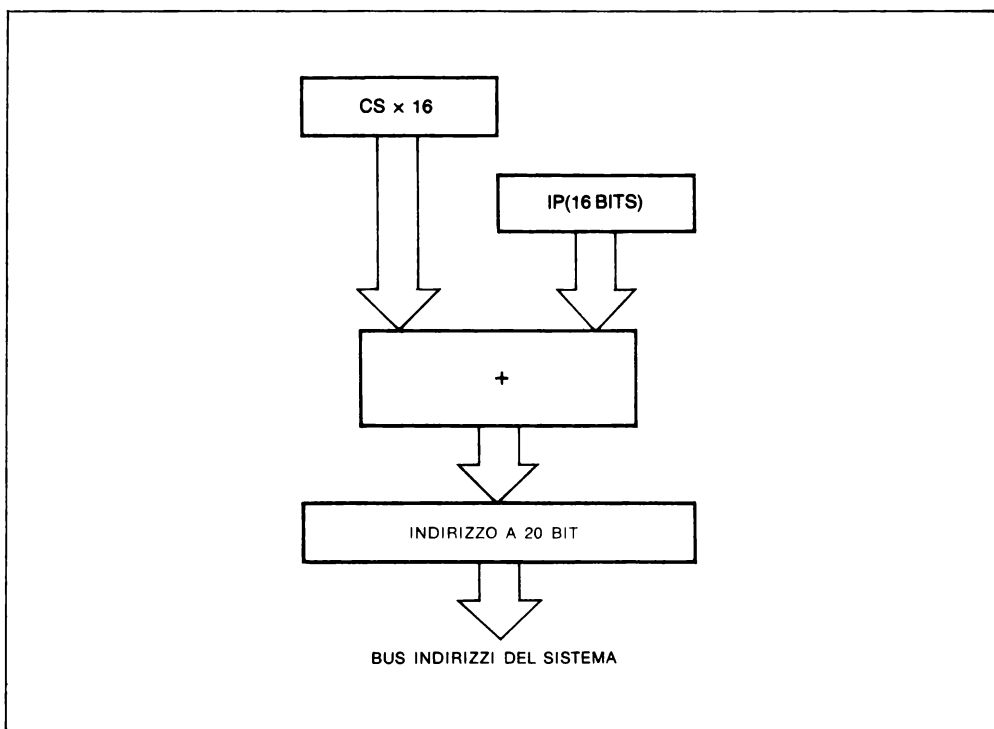



Figura 3.7 — L'indirizzo di sistema a 20 bit di un'istruzione viene calcolato usando l'equazione (CSx16) + IP.

CS x 16 indica l'indirizzo di partenza, o segmento, della memoria da cui l'offset è calcolato. Cioè indica l'indirizzo di partenza del segmento di memoria che contiene i codici delle istruzioni; da qui il nome di registro del segmento codice. La Figura 3.7 mostra il diagramma a blocchi, illustrante il metodo di calcolo dell'indirizzo di 20 bit di un'istruzione.

Ogni indirizzo generato dall'8086/8088 usa uno dei 4 registri segmento. Questo registro segmento è poi spostato (shiftato) a sinistra di 4 bit prima di essere addizionato al valore dell'offset. Il valore dell'offset è calcolato mediante l'addizione dei registri interni. L'istruzione nella CPU specifica quali registri interni sono usati per generare l'offset. Notate che un registro segmento non è mai usato nel calcolo di un valore di offset per la generazione di un indirizzo.

Calcolo del valore di offset

Dopo questa breve introduzione sulla formazione di un indirizzo di 20 bit esaminiamo alcuni esempi dei diversi modi di indirizzamento usati. Useremo l'istruzione MOV dell'8086/8088 perchè è di immediata comprensione. Per ora la nostra meta è quella di approfondire la nostra conoscenza dei diversi modi di indirizzamento dell'8086/8088.

L'istruzione MOV

L'istruzione MOV trasferisce un byte o un parola dall'operando sorgente all'operando destinazione (un operando è un dato che deve essere elaborato dalla CPU). L'istruzione è scritta nel seguente modo:

MOV destinazione, sorgente.

Indirizzamento immediato

Nel modo di indirizzamento immediato l'operando appare nella istruzione. Una istruzione immediata è ad esempio quella che muove un valore costante in un registro interno (che in questo esempio è AX):

MOV AX,568

La costante 568 è caricata nel registro interno AX. Il numero 568 risiederà nei byte di memoria che contengono l'istruzione.

Indirizzamento a registro

Il modo di indirizzamento a registro indica che l'operando che deve essere usato è contenuto in uno dei registri interni generali della CPU. Nel caso dei registri AX, BX, CX, o DX questo registro può essere di 8 o 16 bit. (Questi registri sono stati trattati nel Capitolo 2). Ad esempio, l'istruzione MOV AX,BX copia il contenuto del registro BX nel registro AX. L'istruzione simile MOV AL,BL muove il contenuto del registro BL (8 bit) in AL. Quando si usa l'indirizzamento a registro la CPU effettua tutte le operazioni internamente, così non è necessario generare alcun indirizzo di 20 bit per specificare l'operando.

Esecuzione "interna" di un'istruzione

Indirizzamento diretto

Il modo di indirizzamento diretto specifica completamente nell'istruzione la locazione di memoria che contiene l'operando. In questo tipo di indirizzamento deve essere formato un indirizzo di 20 bit. Questo significa che all'interno dell'8086/8088 avverrà qualche generazione di indirizzo. Ecco un esempio di questo tipo di indirizzamento:

```
MOV CX,COUNT
```

Specificazione del segmento: registro di "default" e prefisso di "override"

Il valore di COUNT è una costante. È usato come valore di offset nel calcolare un indirizzo di 20 bit. L'8086/8088 usa sempre un registro di segmentazione quando calcola un indirizzo fisico. Quale sarà il registro usato per questa particolare istruzione? In mancanza di direttive specifiche sarà usato il registro DS o registro del segmento dati, (vedere Figura 3.8). DS è il segmento di default, cioè quello usato se non ne vengono specificati altri. Tuttavia uno qualsiasi dei 4 valori dei registri di segmentazione può essere usato, purchè nell'istruzione venga specificato il nome del registro da utilizzare.

Per esempio, supponendo di voler usare il registro ES piuttosto che DS dovremmo specificare:

```
MOV CX,ES:COUNT
```

Il valore di ES sarà usato come valore del registro di segmentazione per calcolare l'indirizzo fisico di 20 bit e si dice che ES è il prefisso di override.

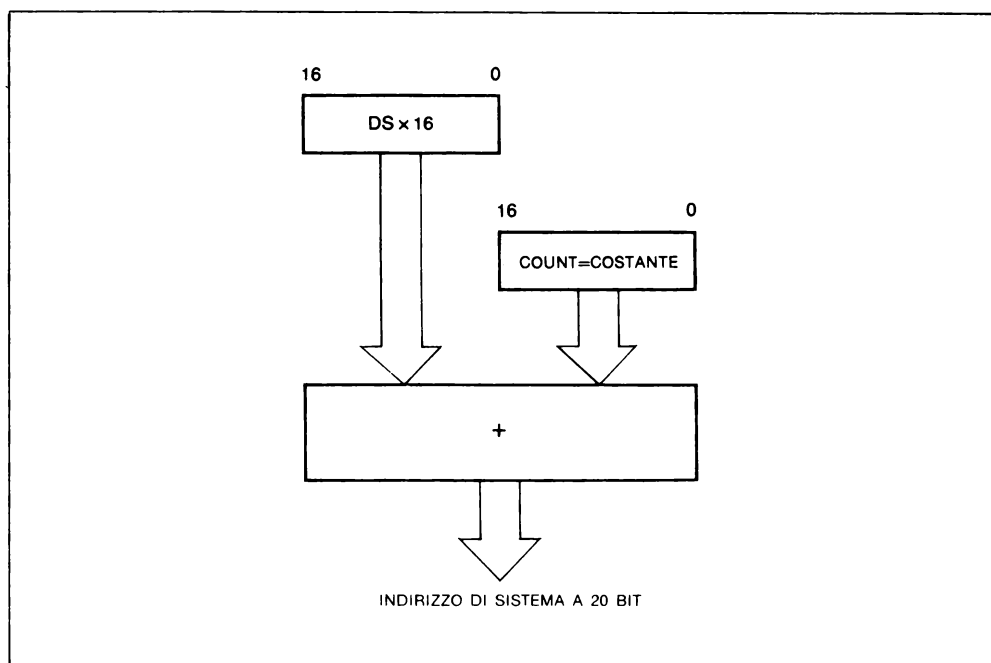



Figura 3.8 — Quando si accede ai dati, il registro DS viene usato con un offset per calcolare l'indirizzo di sistema a 20 bit. DS è il segmento di default È comunque possibile, specificandolo all'interno dell'istruzione, usare un registro diverso da DS.

Indirizzamento indiretto a registro

Con il modo di indirizzamento a registro indice l'offset a 16 bit è contenuto in un registro di base o in un registro indice. Cioè l'indirizzo risiede nel registro BX o in BP o in SI o in DI. Ecco un esempio di questo tipo di indirizzamento:

MOV AX,[SI]

Il valore binario di 16 bit contenuto nel registro SI è l'offset usato per calcolare l'indirizzo di 20 bit. Di nuovo, sarà usato un registro di segmentazione per generare l'indirizzo finale. Il valore a 16 bit contenuto in SI è combinato con il valore dell'appropriato segmento per consentire la generazione dell'indirizzo.

Indirizzamento indiretto a registro con spostamento

Questo tipo di indirizzamento utilizza i due precedenti modi di indirizzamento. L'offset di 16 bit è calcolato sommando il

valore di 16 bit contenuto in un registro interno e una costante. Ad esempio, possiamo usare DI come registro interno e il valore costante COUNT come spostamento, dove COUNT è stato precedentemente definito come un numero. I nomi mnemonici per questo tipo di indirizzamento sono:

MOV AX,COUNT [DI]

Se COUNT = 0378H e DI = 04FAH, allora l'offset a 16 bit sarà 0872H, cioè la somma di 0378H e 04FAH.

Indirizzamento indiretto a registro con base e registro indice

Questo modo di indirizzamento usa la somma di due registri interni dell'8086/8088 per ottenere l'offset a 16 bit che deve essere usato nel calcolo dell'indirizzo a 20 bit. Ecco alcuni esempi di questo tipo di istruzioni:

A MOV,[BP] [DI],AX
B MOV AX,[BX] [SI]

In questa istruzione l'offset A è uguale a BP + DI e l'offset B è uguale a BX + SI.

Indirizzamento indiretto a registro con base + indice + costante

Questo è il più complesso modo di indirizzamento ma se lo si prende un poco alla volta non è troppo difficile. Questo modo di indirizzamento è identico al precedente tranne che per una costante aggiunta alla somma finale di 16 bit. Ad esempio, supponiamo che siano dati i seguenti valori dei registri:

DI = 0367H
BX = 7890H
COUNT = 0012H

Questo modo di indirizzamento indica che l'offset specificato dalla somma di DI + BX + COUNT è usato per spostare i dati dalla memoria nel registro AX. L'istruzione sarebbe scritta nel modo seguente:

MOV AX, COUNT [BX] [DI]

L'indirizzo di offset a 16 bit sarebbe uguale a 7C09H, che è

la somma di 0367H, 7890H e 0012H. L'indirizzo completo di 20 bit è dato dal valore dall'equazione (16 × DS + 7C09H. Se il registro DS è uguale a 3000H allora l'indirizzo completo fisico a 20 bit sarà uguale a (16 × 3000H) + 7C09H = 37C09H

CODICE OGGETTO DELL'8086/8088

I precedenti modi di indirizzo sono quelli usati dall'8086/8088, come programmatori ne scriverete i nomi mnemonici. Il codice oggetto è molto spesso generato dal calcolatore. (Ricordate che il codice oggetto è l'insieme dei byte reali di dati che la CPU esegue). Nel resto di questo capitolo discuteremo in dettaglio come il codice oggetto è generato per le CPU 8086/8088. Queste informazioni vi aiuteranno nel vostro studio dell'insieme di istruzioni dell'8086/8088 (date nel Capitolo 4).

Ora discuteremo i punti più importanti della generazione del codice oggetto dell'8086/8088. Vediamo alcuni esempi; questi esempi sono gli stessi che abbiamo presentato precedentemente nel paragrafo dedicato ai modi di indirizzamento. In ciascun caso presenteremo il codice oggetto e una spiegazione del perchè i bit sono posti ad un 1 logico od a uno 0 logico.

Prima di esaminare qualsiasi esempio di codice oggetto dobbiamo far notare che è difficile dire esattamente quanti byte richiederà una certa istruzione (quelli che hanno precedentemente programmato microprocessori a 8 bit sanno che questo non sempre è vero). Con l'insieme di istruzioni dell'8086/8088 ciascun modo di indirizzamento può richiedere un differente numero di byte. Perciò negli esempi che seguiranno, noi forniremo il numero di byte richiesti per ciascun modo di indirizzamento per una particolare istruzione. Il vostro compito sarà quello di decidere quali devono essere i dati in ciascun byte. Questo compito vi sarà più chiaro man mano che procederemo nella discussione.

Campo REG e bit W

Il primo esempio che tratteremo è l'istruzione MOV AX,568.

Questa istruzione indica uno spostamento di un dato immediato in un registro interno. Il registro interno è la destinazione e può essere un registro lungo un byte o un registro lungo una parola.

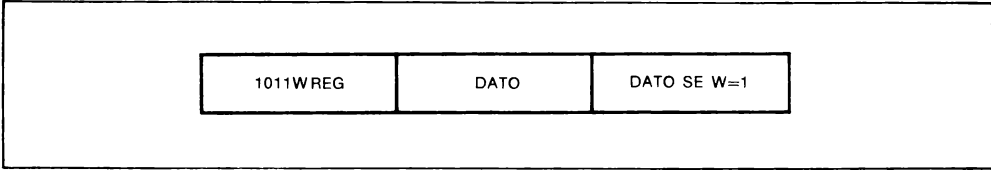



Figura 3.9 — Questa figura mostra il codice oggetto per l'istruzione MOV reg, costante.

Quest'istruzione richiede 2 o 3 byte come mostra la Figura 3.9.

In questa figura possiamo vedere che il primo byte di dati per quest'istruzione ha i 4 bit superiori uguali a 1011.

Significato del bit W

Il bit seguente è il bit W dove W indica una parola (word) se contiene 1 oppure un byte se contiene 0. Cioè, se il registro di destinazione è un registro a 16 bit, o registro parola, allora questo bit è un 1 logico. Se invece il registro destinazione è un registro ad 8 bit, o registro byte, allora questo bit è uno 0 logico.

Funzione del campo REG

I successivi 3 bit del primo byte mostrato in Figura 3.9 sono chiamati REG. Questo campo di 3 bit determina in quale registro, byte, o parola i dati devono essere immessi. I valori dei 3 bit e i relativi registri che essi indicano sono mostrati in Figura 3.10.

Posizione del dato immediato all'interno di un'istruzione

Esaminando i 2 byte seguenti dell'istruzione in Figura 3.9 possiamo vedere che questi sono chiamati DATO. Se il registro di destinazione è un byte allora il dato sta nel secondo byte dell'istruzione. Se la destinazione è una parola allora il

	REGISTRO A 16 BIT	REGISTRO A 8 BIT
000	AX	AL
001	CX	CL
010	DX	DL
011	BX	BL
100	SP	AH
101	BP	CH
110	SI	DH
111	DI	BH

Figura 3.10 — Questa figura dà le definizioni dei registri (a 8 e a 16 bit) per il campo REG del codice oggetto.

secondo byte dell'istruzione è il byte inferiore del dato immediato di 16 bit, mentre il terzo byte dell'istruzione è il byte superiore del dato immediato di 16 bit.

Per quest'esempio (MOV AX,568) assumiamo che il 568 sia un dato immediato esadecimale. Supponiamo inoltre che l'istruzione sia di 3 byte e che i byte siano B8 68 05. Si noti che il primo byte ha il bit W uguale a 1 e il campo REG uguale a 000 per specificare che si usa AX.

Tuttavia, se l'istruzione fosse stata MOV AL, 56 la sua lunghezza sarebbe stata di 2 byte, e questi 2 byte sarebbero stati BO 56. Il primo byte avrebbe avuto il bit W uguale a 0 e il registro uguale a 000 per indicare AL, come mostra la Figura 3.10.

BIT D, MOD E R/M

In questo esempio noi muoveremo i dati dalla memoria e muoveremo un registro da o verso un altro registro. Useremo un'istruzione come MOV AX,BX. Quest'istruzione è di 2 byte perchè non c'è nessun indirizzo di memoria. I byte appariranno come mostra la Figura 3.11.

Possiamo vedere nella Figura 3.11 che i due bit inferiori del primo byte sono DW. Il bit W specifica una parola se vale 1, o byte se vale 0 (come mostrato precedentemente). Il bit D indica se il dato deve essere memorizzato nell'operando specificato dai campi MOD e R/M (in questo caso D = 0), o se deve essere memorizzato nel registro specificato dal campo REG (in questo caso D = 1).

Funzione del bit D

La Figura 3.12 mostra gli assegnamenti a MOD ed R/M. Si noti nella descrizione di MOD che se il suo valore è 11 allora il campo R/M è codificato con il formato di un registro. Questo formato è stato mostrato in Figura 3.10.

In questa istruzione desideriamo immagazzinare il dato nel registro AX, perciò il bit D sarà uno 0 logico. Questo significa che i dati devono essere memorizzati nella locazione specificata dai campi MOD e R/M. Perciò il campo MOD sarà uguale a 11. Il campo R/M sarà uguale a 000 per specificare che la destinazione del dato è il registro AX. Il campo REG sarà uguale a

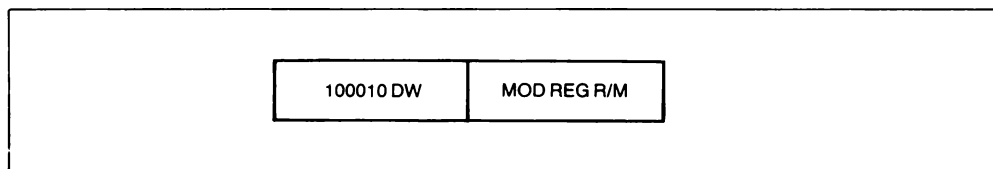



Figura 3.11 — I due byte di un'istruzione di trasferimento di un dato da un registro ad un altro.

REGISTRI BASE ED INDICE SPECIFICATI
DA R/M PER OPERANDI IN MEMORIA
(MOD≠ 11)

CAMPO R/M	REGISTRO BASE	REGISTRO INDICE
000	BX	SI
001	BX	DI
010	BP	SI
011	BP	DI
100	NESSUNO	SI
101	NESSUNO	DI
110	BP	NESSUNO
111	BX	NESSUNO

MOD	SPOSTAMENTO	COMMENTO
00	ZERO	
01	CONTENUTO DI 8 BIT DEL BYTE SUCCESSIVO DELL'ISTRUZIONE. ESTESO A 16 BIT DAL SEGNO	L'ISTRUZIONE CONTIENE UN BYTE ADDIZIONALE
10	CONTENUTO DI 16 BIT DEI 2 BYTE SUCCESSIVI DELL'ISTRUZIONE	L'ISTRUZIONE CONTIENE 2 BYTE ADDIZIONALI
11	REGISTRO R/M	

SE MOD = 00 ED R/M = 110. ALLORA
1. LE INDICAZIONI DATE NON SONO VALIDE
2. L'ISTRUZIONE CONTIENE 2 BYTE ADDIZIONALI
3. L'OFFSET È CONTENUTO IN QUEI DUE BYTE

Figura 3.12 — Definizioni del campo R/M e di MOD per il codice oggetto dell'8086/8088.

011, in tal modo si indica che deve essere usato il registro BX come registro sorgente (questo valore deriva dalla Figura 3.10). Il valore completo del secondo byte dell'istruzione sarà 11011000 o D8. Perciò il codice oggetto dell'istruzione MOV AX,BX è 89D8.

CODICE OGGETTO PER L'INDICIZZAZIONE E USO DEL REGISTRO DI BASE

Esaminiamo un ultimo esempio di generazione del codice oggetto per l'8086/8088. In questo esempio calcoleremo il codice oggetto per l'istruzione MOV CX, COUNT [BX] [SI]. Questa istruzione ha lo stesso formato generàle dell'esempio precedente. Il numero di byte necessari per questa istruzione è 4. Questi byte sono mostrati nella Figura 3.13.

Il primo byte della Figura 3.13 ha il bit D posto uguale ad un 1 logico. Questo perchè la destinazione dei dati è specificata dal campo REG nel secondo byte. Il bit W è un 1 logico perchè viene trasferita una parola. Il contenuto del primo byte è quindi uguale a 10001011 o 8B.

Osserviamo ora il campo MOD del secondo byte. Poichè stiamo usando una costante che richiede 16 bit, MOD sarà uguale a 10. Con riferimento alla Figura 3.12 ciò indica che lo spostamento sarà formattato in 2 byte e seguirà questo byte.

Il campo seguente del secondo byte è il campo registro o REG. Poichè usiamo il registro CX, questo valore è uguale a 001 (questo valore è stato ottenuto dalle definizioni del campo REG date in Figura 3.10).

Infine, abbiamo il campo R/M. Poichè il campo MOD non era uguale a 11, R/M specifica quali registri di base e di indice devono essere usati per generare un offset di 16 bit. Nel nostro caso stiamo usando il campo [BX+SI+SPOST]. Questo corrisponde a un valore di R/M uguale a 000. (Vedere Figura 3.12).

Il valore completo del secondo byte di dati sarà 10001000, corrispondente a 88 in base 16. Il terzo e quarto byte saranno uguali allo spostamento. Nel nostro caso il valore COUNT sarà uguale a 0345 in esadecimale. Gli ultimi 2 byte saranno 4503. Questo porta ad un codice oggetto completo per l'istruzione MOV CX,COUNT [BX] [SI]   8B884503.

SOMMARIO DEL CODICE OGGETTO

Una domanda che si può fare a questo punto è "Devo conoscere il valore di D, W, REG, MOD, e R/M ogni istante?" Se così fosse probabilmente non programmereste l'8086/8088.

Fortunatamente il vostro calcolatore vi aiuterà a calcolari.

Questo paragrafo dedicato al codice oggetto è stato presentato solo per permettere di capire meglio i meccanismi interni dei microprocessori 8086/8088.

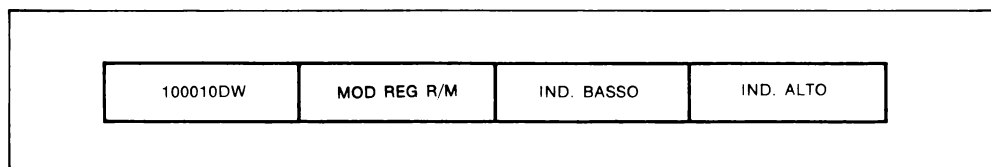



Figura 3.13 — Il formato del codice oggetto per un'istruzione come MOV BX; COUNT [BX] [SI].

SOMMARIO

In questo capitolo abbiamo discusso alcuni argomenti importanti relativi alla programmazione dei microprocessori 8086/8088. Abbiamo cominciato con la presentazione dell'organizzazione della memoria del sistema per entrambe le CPU. Questa discussione è stata fatta per aiutarvi a capire più facilmente come le istruzioni sono organizzate nella memoria del sistema; inoltre, vi permette di vedere le differenze e le somiglianze fra i sistemi 8086/8088.

Abbiamo poi discusso tutti i modi di indirizzamento.

Durante la presentazione di ciascun modo sono stati datdati esempi per mostrare esattamente come sono stati generati gli indirizzi. Abbiamo anche discusso i registri di segmentazione, di base e di indice e abbiamo imparato come un indirizzo fisico di sistema di 20 bit viene generato internamente dalla CPU.

Abbiamo concluso il capitolo con dègli esempi che mostrano come generare il codice oggetto per l'8086/8088.

Questi esempi comprendono una discussione dei diversi campi che formano il codice oggetto, compresi i campi D, W, R/M, REG e MOD.

Le informazioni date in questo capitolo dovrebbero aiutarvi a capire meglio la codifica delle istruzioni mostrate nel prossimo capitolo.

CAPITOLO 4

IL SET DI ISTRUZIONI: DESCRIZIONI INDIVIDUALI

Presenteremo in questo capitolo l'insieme completo delle istruzioni dell'8086/8088, in ordine alfabetico rispetto ai nomi mnemonici. Daremo una breve descrizione di ogni istruzione e mostreremo l'uso in linguaggio assembly di ognuna di esse (inclusi gli argomenti). Inoltre, descriveremo come l'8086/8088 esegue ogni istruzione. Questo è utile se volete un riferimento rapido che mostri ciò che succede realmente durante l'esecuzione di un'istruzione. Ogni descrizione include qualsiasi considerazione speciale che potrebbe essere necessaria per una particolare istruzione e specifica quando un'istruzione può o non può essere usata.

Per certe istruzioni faremo degli esempi. Per altre sarà chiaro dalle descrizioni come devono essere usate esattamente.

Infine, mostreremo il codice oggetto per ogni istruzione.

I nomi mnemonici usati in questo capitolo sono coerenti con quelli usati dalla Intel Corporation. Le informazioni presentate in questo capitolo dovrebbero non solo aiutarvi a capire meglio come può essere programmato il microprocessore 8086/8088, ma dovrebbero anche servirvi da prezioso riferimento quando comincerete veramente a scrivere programmi in linguaggio assembly.

La tavola qui sotto mostra le abbreviazioni ed i simboli, e la Figura 4.1 mostra il modello dei registri dell'8086/8088 che useremo nelle descrizioni che seguiranno le istruzioni.

ABBREVIAZIONI E SIMBOLI PER LA DESCRIZIONE DELLE ISTRUZIONI

Se d = 1, allora "to"; se d = 0, allora "from"
Se w = 1, allora parola; se w = 0, allora byte
Se s:w = 01, allora 16 bit dei dati immediati formano l'operando
Se s:w = 11, allora un byte di dati immediati è esteso a 16 bit dal segno per formare l'operando

(segue)

Se v = 0, allora "count" = 1, se v = 1, allora "count" in CL
x = indifferente
z è usato per alcune stringhe di istruzione da confrontare con il flag ZF
AL = accumulatore a 8 bit
AX = accumulatore a 16 bit
CX = registro contatore
DX = registro porta variabile
DS = segmento dati
ES = segmento extra
Superiore/inferiore riferito a valore senza segno

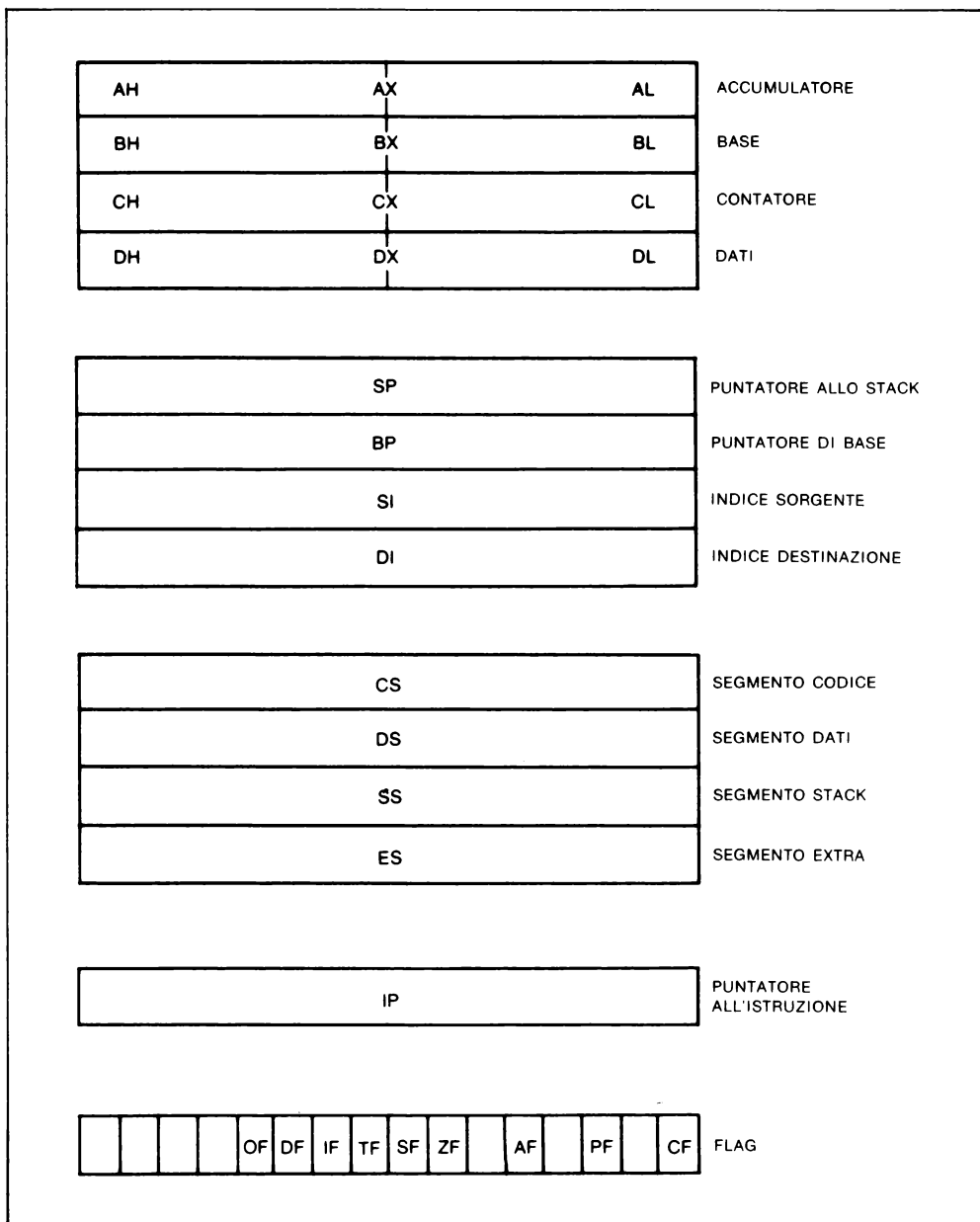



Figura 4.1 — I registri interni dell'8086/8088.

AAA — Adattamento ASCII per l'addizione

Nome mnemonico:

AAA nessun operando

Funzione:

Se D3, D2, D1, D0 di AL>9 o flag AF=1, allora AL=AL+6, AH=AH+1, AF=1, CF=AF, AL=(AL "and" OFh)

Flag definiti:

AF, CF

Flag indefiniti:

OF, PF, SF, ZF

Descrizione:

L'istruzione AAA cambia il valore del registro AL in un numero decimale non impaccato valido; i bit D7—D4 del registro AL sono azzerati.

Esempio:

AAA

Prima:	*Dopo:*
AL = 0Bh	AL = 01h
AH = 00h	AH = 01h

Codifica:

00110111

— Adattamento ASCII per la divisione — AAD

Nome mnemonico:

AAD nessun operando

Funzione:

AL=((AH*0Ah)+AL;AH=0)

Flag definiti:

PF, SF, ZF

Flag indefiniti:

AF, CF, OF

Descrizione:

L'istruzione AAD cambia i contenuti dei registri AL e AH (ciascuno dei quali contiene un numero BCD non impaccato valido) in un numero binario equivalente al contenuto nel registro AL. Questo permette al programmatore di usare l'istruzione IDIV per ottenere il risultato corretto. Il registro AH deve essere zero prima dell'istruzione IDIV. Dopo l'istruzione IDIV, il quoziente è posto in AL ed il resto è posto in AH. Entrambi i semibyte di ordine alto vengono azzerati.

Esempio:

AAD

Prima:	*Dopo:*
AL = 03h	AL = 35h
AH = 05h	AH = 00H

Codifica:

11010101	00001010

AAM — Adattamento ASCII per la moltiplicazione

Nome mnemonico:

AAM nessun operando

Funzione:

AH=AL/0Ah; AL=resto

Flag definiti:

PF, SF, ZF

Flag indefiniti:

AF, CF, OF

Descrizione:

L'istruzione AAM corregge il Risultato di una moltiplicazione fra due numeri BCD non impaccati validi. Un numero BCD non impaccato valido è formato dal contenuto di AH e AL. Il numero binario in AL è diviso per 10 ed il quoziente viene immagazzinato nel registro AL. I semibyte di ordine alto degli operandi moltiplicati dovevano essere 0h affinchè AAM potesse dare un risultato corretto.

Esempio:

AAM

Prima:	*Dopo:*
AH = 00h	AH = 06h
AL = 41h	AL = 05h

Codifica:

11010100	00001010

Adattamento ASCII per la sottrazione — AAS

Nome mnemonico:

AAS nessun operando

Funzione:

Se D3, D2, D1, D0 di AL>9 o flag AF=1, allora AL=AL–6, AH=AH+1, AF=1, CF=AF, AL=(AL "and" OFh)

Flag definiti:

AF, CF

Flag indefiniti:

OF, PF, SF, ZF

Descrizione:

AAS corregge il risultato di una precedente sottrazione di due operandi decimali non impaccati validi. AAS cambia il contenuto di AL in un numero decimale non impaccato valido. Il semibyte di ordine alto viene azzerato.

Codifica:

00111111

ADC — Addizione con riporto

Nome mnemonico:

ADC destinazione, sorgente

Funzione:

Se CF=1 allora destinazione = destinazione + sorgente + 1

Se CF=0 allora destinazione = destinazione + sorgente

Flag definiti:

AF, CF, OF, PF, SF, ZF

Descrizione:

ADC somma gli operandi che possono essere byte o parole, poi addiziona 1 se CF vale 1 e sostituisce l'operando destinazione con il risultato. Entrambi gli operandi possono essere numeri binari con o senza segno.

Flag definiti:

PF, SF, ZF

Flag indefiniti:

AF, CF, OF

Esempio:

```
ADC AX,BX
ADC AL,9
ADC BX,458
```

Codifica:

Somma l'operando in un registro o in memoria con l'operando in un registro

000100dw	mod reg r/m

Somma l'operando immediato all'operando in un registro o in memoria

100000sw	mod 010 r/m	dato	dato se s:w=01

Somma l'operando immediato all'accumulatore

0001010w	dato	dato se w=1

Nome mnemonico:

ADD destinazione, sorgente

Funzione:

Destinazione = destinazione + sorgente

Flag definiti:

AF, CF, OF, PF, SF, ZF

Descrizione:

La somma dei due operandi, che possono essere byte o parole, sostituisce l'operando destinazione.

Esempio:

```
ADD DX,CX
ADD AX,400
ADD label,BL
```

Codifica:

Somma l'operando in un registro o in memoria con l'operando in un registro

000000dw	mod reg r/m

Somma l'operando immediato all'operando in un registro o in memoria

100000sw	mod 000 r/m	dato	dato se s:w=01

Somma l'operando immediato all'accumulatore

0000010w	dato	dato se w=1

AND — And logico

Nome mnemonico

AND destinazione, sorgente

Funzione:

Destinazione = destinazione "and" sorgente
CF=0, OF=0

Flag definiti:

CF, 0F, PF, SF, ZF

Flag indefiniti:

AF

Descrizione:

AND esegue l'"and" logico dei due operandi (byte o parola) e memorizza il risultato nell'operando destinazione. Un bit nel risultato è posto ad un 1 logico se entrambi i bit degli operandi originali sono 1 logici; altrimenti, il bit viene azzerato.

Esempio:

AND BX,CX

Prima:	*Dopo:*
BX=AC07h	BX=2004h
CX=23F4h	CX=23F4h

Codifica:

AND dell'operando in un registro o in memoria con l'operando in un registro

001000dw	mod reg r/m

AND tra l'operando immediato e l'operando destinazione in un registro o in memoria

1000000w	mod 100 r/m	dato	dato se w=1

AND tra l'operando immediato e l'operando destinazione nell'accumulatore

0010010w	dato	dato se w=1

CALL — Chiamata di una Procedura

Nome mnemonico:

CALL Nome della procedura

Funzione:

Se intersegmento allora: SP = SP – 2; il valore attuale di CS è salvato sullo stack
CS = nuovo segmento specificato: SP = SP – 2; IP è salvato sullo stack
IP = nuovo puntatore all'istruzione specificato

Flag influenzati:

Nessuno

Descrizione:

CALL attiva una procedura fuori linea salvando nello stack le informazioni necessarie per permettere ad un'istruzione RET (ritorno) eseguita nella procedura chiamata di ritrasferire il controllo all'istruzione che segue la CALL.

L'assemblatore genera un diverso tipo di istruzione CALL, a seconda che il programmatore abbia definito il nome della procedura come NEAR (vicino) o FAR (lontano). Affinchè il controllo possa ritornare al programma principale in modo corretto, il tipo di istruzione CALL deve essere abbinato all'appropriato tipo di istruzione di ritorno (RET) che provoca l'uscita dalla procedura.

In una CALL intrasegmento diretta, SP viene decrementato di 2 e IP è salvato nello stack. Lo spostamento relativo (fino a ± 32K) della procedura dall'istruzione CALL, cioè la distanza tra l'inizio della procedura chiamata e l'istruzione CALL, viene poi sommato al valore del puntato all'istruzione (IP).

Una CALL intersegmento indiretta, può essere fatta attraverso la memoria o un registro. SP viene decrementato di 2 e IP viene salvato nello stack. L'offset (spostamento) della procedura chiamata viene ottenuto tramite la parola in memoria o il registro generale a 16 bit a cui si fa riferimento nell'istruzione che sostituisce IP.

In una CALL intrasegmento diretta, SP viene decrementato di 2 e CS viene messo nello stack. CS viene sostituito dalla parola segmento contenuta nell'istruzione.

SP viene nuovamente decrementato di 2, e IP viene salvato nello stack e sostituito dalla parola che specifica l'offset e che è anch'essa contenuta nell'istruzione.

In una CALL intersegmento indiretta, SP viene decrementato di 2 e CS viene salvato nello stack. CS viene poi sostituito dal contenuto della seconda parola del puntatore di memoria (lungo due parole) a cui si fa riferimento nell'istruzione. SP viene nuovamente decrementato di 2 ed IP viene salvato nello stack e sostituito dal contenuto della prima parola del puntatore a cui si fa riferimento nell'istruzione.

Esempio:

CALL NEAR
CALL FAR
CALL AX

Codifica:

Chiamata intrasegmento diretta

11101000	spost-basso	spost-alto

destinazione = indirizzo effettivo

Chiamata intrasegmento indiretta

11111111	mod 010 r/m

destinazione = IP + spostamento

Chiamata intersegmento diretta

100110100	offset-basso	offset-alto	seg-basso	seg-alto

destinazione = offset, segmento = segmento

Chiamata intersegmento indiretta

11111111	mod 011 r/m

destinazione = indirizzo effettivo, segmento = indirizzo effettivo + 2

CBW — Conversione di un byte in una parola —

Nome mnemonico:

CBW nessun operando

Funzione:

Se AL<80h allora AH=00h
Se AL > 80h allora AH = FFh

Flag influenzati:

Nessuno

Descrizione:

CBW estende a tutto il registro AH il segno del byte contenuto nel registro AL.

Esempio:

CBW

Prima:	*Dopo:*
AL = 83h	AH = FFh: AL = 83h
AL = 56h	AH = 00h: AL = 56h

Codifica

10011000

CLC

Nome mnemonico:

CLC Nessun operando

Funzione:

CF = 0

Flag definiti:

CF

Descrizione:

CLC azzera il flag del riporto (CF) e non influenza nessun altro flag.

Codifica

11111000

CLD — Azzeramento del flag di direzione

Nome mnemonico:

CLD Nessun operando

Funzione:

DF = 0

Flag definiti:

DF

Descrizione:

CLD azzera DF, causando un autoincremento del registro indice destinazione DI e/o del registro indice sorgente SI quando vengono eseguite istruzioni che trattano stringhe.

Codifica

11111100

— Azzeramento del flag di abilitazione delle interruzioni — CLI

Nome mnemonico:

CLI

Funzione:

IF = 0

Flag definiti:

IF

Descrizione:

CLI azzera il flag IF, disabilitando le interruzioni mascherabili. Le interruzioni software non vengono disabilitate.

Codifica:

11111010

CMC — Complementazione del flag di riporto

Nome mnemonico:

CMC Nessun operando

Funzione:

Se CF = 0 allora CF = 1; se CF = 1 allora CF = 0

Flag definiti:

CF

Descrizione:

CMC trasforma CF nel suo stato opposto

Codifica:

11110101

Nome mnemonico:

CMP destinazione, sorgente

Funzione:

Destinazione – sorgente

Flag definiti:

AF, CF, OF, PF, SF, ZF

Descrizione:

CMP sottrae l'operando sorgente dall'operando destinazione, che possono essere byte o parole, ma non memorizza il risultato. Gli operandi non vengono cambiati, l'effetto dell'istruzione è quello di aggiornare i valori dei flag, i quali possono essere testati con una successiva istruzione di salto condizionato.

Esempio:

```
CMP DX,CX
CMP AL,25
CMP BH,Label
```

Codifica:

Confronto tra un operando in un registro o in memoria e un operando in un registro

001110dw	mod reg r/m

Confronto tra un operando immediato e un operando in un registro o in memoria

100000sw	mod 111 r/m	dato	dato se s:w=01

Confronto tra un operando immediato e un operando nell'accumulatore

0011110w	dato	dato se w=1

CMPS — Confronto fra stringhe (di byte o parole)

Nome mnemonico:

CMPS (stringa-destinazione) – (stringa-sorgente)

Funzione:

(stringa-destinazione) – (stringa-sorgente)
Se DF = 0, allora SI = SI + delta, DI = DI + delta
Se DF = 1, allora SI = SI – delta, DI = DI – delta

Se la stringa è un byte, allora delta =1
Se la stringa è una parola allora delta =2

La stringa destinazione è indirizzata da DI
La stringa sorgente è indirizzata da SI

Flag definiti:

AF, CF, OF, PF, SF, ZF

Descrizione:

CMPS (confronto fra stringhe) sottrae il byte o la parola sorgente (indirizzata da DI) dal byte o parola destinazione (indirizzata da SI). CMPS influenza i flag ma non altera alcun operando. Se CMPS viene preceduta da REPE o REPZ, l'operazione viene interpretata come "confronta fino a quando non si è alla fine della stringa (CX non è 0) e le stringhe sono uguali (ZF =1)". Se CMPS è preceduto da REPNE o REPNZ l'operazione va intesa come "confronta fino a quando non si è alla fine della stringa (CX non è 0) e le stringhe non sono uguali (ZF =0)". Perciò CMPS può essere usato per trovare gli elementi delle stringhe uguali o differenti.

Gli operandi nominati nell'istruzione CMPS vengono usati solo dall'assembler per verificare il tipo e l'accessibilità usando i contenuti attuali dei registri di segmentazione.

Codifica:

1010011w

Conversione di una parola in una parola doppia — CWD

Nome mnemonico:

CWD nessun operando

Funzione:

Se AX < 8000h allora DX =0
Se AX > 8000h allora DX =FFFFh

Flag influenzati:

Nessuno

Descrizione:

CWD estende a tutto il registro DX il segno della parola contenuta nel registro A

Esempio:

CWD

Prima:	*Dopo:*
AX = 9034h	DX = FFFFh, AX = 9034h
AX = 7034h	DX = 0000h, AX = 7034h

Codifica:

10011001

DAA — Adattamento decimale per l'addizione —

Nome mnemonico:

DAA nessun operando

Funzione:

Se D3, D2, D1, D0 di AL >9 o flag AF =1,
allora AL = AL +6: AF=1
Se AL > 9Fh o flag CF =1,
allora AL = AL +60h: CF=1

Flag definiti:

AF, CR, PF, SF, ZF

Flag indefiniti:

OF

Descrizione:

DAA cambia il contenuto di AL in una coppia di cifre decimali impaccate valide.

Esempio:

DAA

Prima:	*Dopo:*
AL = 8Ah	AL = 90h

Codifica:

00100111

Adattamento decimale per la sottrazione

Nome mnemonico:

DAS nessun operando

Funzione:

Se D3, D2, D1, D0 di AL >9 o AF =1,
allora AL = AL –6; F=1
Se AL > 9Fh o CF =1 allora AL = AL –60h; CF=1

Flag definiti:

AF, CR, PF, SF, ZF

Flag indefiniti:

OF

Descrizione:

DAS cambia il contenuto di AL in una coppia di cifre decimali impaccate valide che erano il risultato di una precedente sottrazione.

Esempio:

DAS

Prima:	*Dopo:*
AL = 1Fh	AL = 19h

Codifica:

00101111

DEC — Decremento

Nome mnemonico:

DEC destinazione

Funzione:

Destinazione = destinazione − 1

Flag definiti:

AF, OF, PF, SF, ZF

Descrizione:

DEC sottrae 1 dall'operando destinazione. L'operando può essere un byte o una parola.

Esempio:

DEC BX

Prima:	*Dopo:*
BX = 12h	BX=11h

Codifica:

Decremento di un operando in un registro o in memoria

1111111w	mod 001 r/m

Decremento di un operando in un registro

01001reg

Nome mnemonico:

DIV sorgente

Funzione:

Se l'operando sorgente è un byte: AX/sorgente: AH = quoziente AL = resto

Se l'operando sorgente è una parola: AX,DX/sorgente, AX, = quoziente, DX = resto, in cui sono contenuti i 16 bit più significativi per la divisione.

Flag definiti:

Nessuno

Flag indefiniti:

AF, CF, OF, PF, SF, ZF

Descrizione:

DIV (divisione) esegue una divisione non segnata dell'accumulatore (e della sua estensione) per l'operando sorgente. Se l'operando sorgente è un byte, esso viene diviso nel dividendo a doppia lunghezza che si assume sia nei registri AL e AH. Il resto a lunghezza singola viene ritornato in AH. Se l'operando sorgente è una parola, essa viene divisa nel dividendo a doppia lunghezza nei registri AX e DX. Il resto a lunghezza singola viene ritornato in DX. Se il quoziente va oltre la capacità del proprio registro di destinazione (FFH per la sorgente di un byte, FFFFH per la sorgente di una parola), e quando si tenta la divisione per zero si genera un'interruzione di tipo 0. In questo caso il quoziente ed il resto sono indefiniti. I quozienti non interi vengono troncati ad interi.

Esempio:

DIV 25 (questo è un byte)

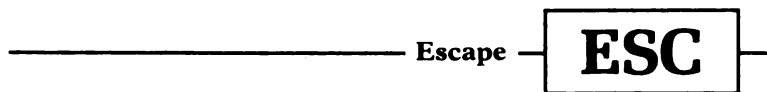
Prima:	*Dopo:*
AH = 00h	AH = 01h resto
AL = 33h	AL = 02h quoziente

DIV 300 (questa è una parola)

Prima:	*Dopo:*
DX = 0000h	DX = 0005h resto
AX = 0389h	AX = 0003h quoziente

DIV CL
DIV CX

Codifica:

1111011w	mod 110 r/m	possono seguire 2-4 byte

Nome mnemonico:

ESC codice operativo esterno, indirizzo

Funzione:

Se MOD non è uguale a 11, l'istruzione viene messa sul bus dati.
Se MOD è uguale a 11 è eseguita una non operazione (vedere istruzione NOP).

Flag influenzati

Nessuno

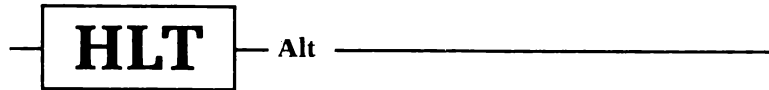
Descrizione:

L'istruzione Escape fornisce un meccanismo tramite il quale altri processori (co-processori) possono ricevere le loro istruzioni dal flusso di istruzioni dell'8086/8088 e possono utilizzare i modi di indirizzamento dell'8086/8088. La CPU non esegue alcuna operazione per l'istruzione ESC eccetto quella di accedere alla memoria e di mettere l'operando sul bus dati. Gli 8086/8088 sono progettati per operare con una varietà di co-processori che eseguono diverse funzioni di sistema. Due coprocessori che operano con l'8086/8088 sono l'8089 Input e Output Processor e l'8087 Numeric Data Processor.

Esempio:

ESC 6, AL

Codifica

11011xxx	mod xxx r/m

Nome mnemonico:

HLT nessun operando

Funzione:

Nessuna

Flag influenzati:

Nessuno

Descrizione:

HLT fa entrare l'8086/8088 nello stato di alt. Il processore abbandona lo stato di alt dopo l'attivazione della linea RESET, dopo aver ricevuto una richiesta di una interruzione non mascherabile su NMI, o, se le interruzioni sono abilitate, dopo aver ricevuto la richiesta di una interruzione mascherabile su INTR.

Codifica:

11110100

Nome mnemonico:

IDIV sorgente

Funzione:

Se l'operando sorgente è un byte, il quoziente viene memorizzato in AL, il resto viene memorizzato in AH e il dividendo è contenuto in AX e viene diviso per l'operando sorgente indicato.
Se l'operando sorgente è una parola, il quoziente viene memorizzato in AX, il resto viene memorizzato in DX, e il dividendo è contenuto in AX e DX ed è diviso per l'operando sorgente indicato.

Flag definiti:

Nessuno

Flag indefiniti:

AF, CF, OF, PF, SF, ZF

Descrizione:

La divisione intera esegue una divisione con segno dell'accumulatore (e della sua estensione) per l'operando sorgente. Se l'operando sorgente è un byte allora: il dividendo a doppia lunghezza è contenuto nei registri AL ed AH; dopo la divisione del dividendo per l'operando sorgente il quoziente a lunghezza singola è memorizzato in AL ed il resto, anch'esso a lunghezza singola, è memorizzato in AH. Per le divisioni intere di byte, il massimo quoziente positivo è +127 (7FH) e il minimo quoziente negativo è −127 (81H). Se l'operando sorgente è una parola allora: il dividendo a doppia lunghezza è contenuto nei registri AX e DX; dopo la divisione del dividendo per l'operando sorgente, il quoziente a singola lunghezza è memorizzato in AX ed il resto, anch'esso a singola lunghezza, è memorizzato in DX. Per le divisioni intere di parole, il massimo quoziente positivo è +2767 (7FFFH) e il minimo quoziente negativo è −32767 (8001H).

Se il quoziente è positivo ed è maggiore del massimo, o è negativo ed è minore del minimo, il quoziente ed il resto sono indefiniti e si genera un'interruzione di tipo 0. I quozienti non interi vengono troncati (verso 0) ad interi, ed il resto ha lo stesso segno del dividendo.

Esempio:

IDIV CL
IDIV BX

Codifica:

1111011w	mod 111 r/m

Nome mnemonico:

IMUL sorgente

Funzione:

Se l'operando sorgente è un byte: AX=AL × sorgente (con segno)
Se l'operando è una parola: AX,DX=AX × sorgente (con segno)

Flag definiti:

CF, OF

Flag indefiniti:

AF, PF, SF, ZF

Descrizione:

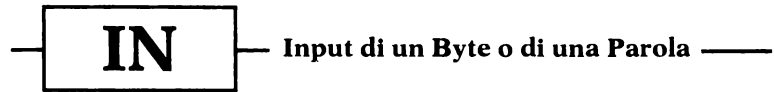
IMUL esegue una moltiplicazione con segno dell'operando sorgente e dell'accumulatore. Se l'operando sorgente è un byte, allora viene moltiplicato per il registro AL ed il risultato a doppia lunghezza viene memorizzato in AH ed AL. Se l'operando sorgente è una parola, allora viene moltiplicato per il registro AX ed il risultato a doppia lunghezza viene memorizzato nei registri DX ed AX. Se la metà superiore del risultato (AH per la sorgente di un byte, DX per la sorgente di una parola) non è l'estensione del segno della metà inferiore del risultato, CF ed OF sono posti ad 1; altrimenti sono azzerati. Quando CF e OF valgono 1 indicano che AH o DX contengono cifre significative del risultato.

Esempio:

IMUL BL
IMUL CL
IMUL 400

Codifica:

1111011w	mod 101 r/m

IN — Input di un Byte o di una Parola

Nome mnemonico:

IN accumulatore, porta

Funzione:

Se byte: AL=dato dalla porta specificata, o AL=dato indirizzato dal registro DX.
Se parola: AX=dato dalla porta specificata, o AX=dato indirizzato dal registro DX

Flag influenzati:

Nessuno

Descrizione:

IN trasferisce un byte o una parola da una porta di input al registro AL o AX, rispettivamente. Il numero della porta può essere specificato o con una costante immediata di un byte, ciò permette l'accesso alle porte 0–255, o con un numero messo precedentemente nel registro DX, e ciò permette l'accesso alle porte 0–65535.

Esempio:

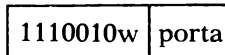
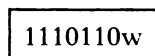
IN AL,045H	byte da una porta specificata in modo immediato
IN AX,046H	parola da una porta specificata in modo immediato
IN AL,DX	byte da una porta variabile
IN AX,DX	parola da una porta variabile

Codifica:

Porta fissa

1110010w	porta

Porta variabile

1110110w

l'indirizzo della porta è contenuto nel registro DX

Nome mnemonico:

INC destinazione

Funzione:

Destinazione = destinazione + 1

Flag definiti:

AF, OF, PF, SF, ZF

Descrizione:

INC aggiunge 1 all'operando destinazione. L'operando può essere un byte o una parola.

Esempio:

INC BL
INC BX

Codifica:

Incremento di un operando in un registro o in memoria

1111111w	mod 000 r/m

Incremento di un operando in un registro

01000reg

Nome mnemonico:

INT tipo dell'interruzione da 0 a 255

Funzione:

SP= – 2; flag salvati sullo stack; IF=0; TF=0; SP=SP – 2; CS salvato sullo stack
Nuovo CS=dati all'indirizzo di memoria (tipo * 4 + 2);
SP=SP – 2; IP salvato sullo stack;
Nuovo IP=dati all'indirizzo di memoria tipo *4.

Flag definiti:

TF, IF

Descrizione:

INT attiva la procedura d'interruzione specificata dall'operando (tipo dell'interruzione). L'indirizzo del puntatore all'interruzione viene calcolato moltiplicando il tipo dell'interruzione per 4. La seconda parola del puntatore all'interruzione sostituisce CS. La prima parola del puntatore all'interruzione sostituisce IP.

Se è specificato il tipo 3, l'Assembler genererà una forma speciale a un byte dell'istruzione di interruzione. Ciò avviene perchè il 3 corrisponde all'interruzione di breakpoint.

Esempio:

INT 3
INT 58

Codifica:

1100110v	tipo se v=1

Se v=0, allora tipo=3

Nome mnemonico:

INTO nessun operando

Funzione:

Se OF=1 allora: SP=SP – 2; i flag vengono salvati sullo stack; IF=0; TF=0; SP=SP – 2; CS viene salvato sullo stack; nuovo CS=dati alla locazione di memoria 12H; SP=SP – 2; IP viene salvato sullo stack; nuovo IP=dati alla locazione di memoria 10H.

Flag influenzati:

TF, IF

Descrizione:

INTO genera un'interruzione software se il flag di overflow (OF) vale 1; altrimenti il controllo passa all'istruzione successiva senza che venga attivata una procedura di interruzione.

Codifica:

11001110

IRET — Ritorno da un'interruzione

Nome mnemonico:

IRET nessun operando

Funzione:

Ad IP è assegnato il valore che si trova al top dello stack; SP=SP + 2; a CS è assegnato il valore che dopo l'incremento di SP si trova al top dello stack; SP=SP + 2; ai flag è assegnato il valore che dopo il nuovo incremento di SP si trova al top dello stack; SP=SP + 2.

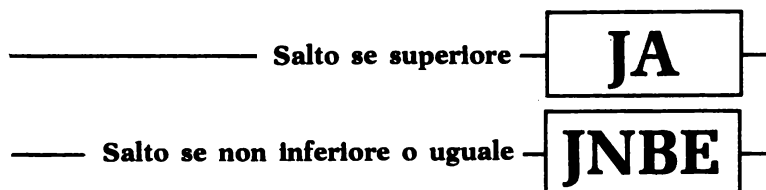
Flag definiti:

Tutti

Descrizione:

IRET ritrasferisce il controllo al punto in cui era avvenuta l'interruzione, prelevando il valore di IP, di CS e dei flag dall'area di stack. IRET viene usato per uscire da qualsiasi procedura di interruzione, attivata sia dall'hardware che dal software.

Codifica:

11001111

Nomi mnemonici:

JA spostamento ad 8 bit con segno
JNBE spostamento ad 8 bit con segno

Funzione:

Se CF e ZF=0, allora IP=IP + spostamento a 8 bit.
Lo spostamento subisce un'estensione del segno a 16 bit.

Flag influenzati:

Nessuno

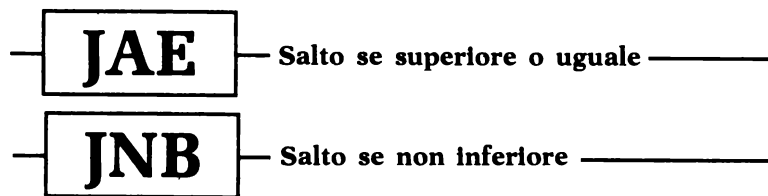
Descrizione:

JA e JNBE trasferiscono il controllo all'operando "obiettivo" (IP + spostamento) a condizione che CF e ZF=0 (o falso). L'assembler genera lo stesso codice per entrambi questi nomi mnemonici.

Esempio:

JA label
JNBE label

Codifica:

01110111	spostamento

Nomi mnemonici:

JAE spostamento ad 8 bit con segno
JNB spostamento ad 8 bit con segno

Funzione:

Se CF=0, allora IP=IP + spostamento (esteso a 16 bits dal segno)

Flag influenzati:

Nessuno

Descrizione:

JAE e JNB trasferiscono il controllo dell'operanddo "obiettivo" (IP + spostamento) se la condizione (CF=0) è superiore o uguale / non inferiore del valore testato. Entrambi questi nomi mnemonici genereranno lo stesso codice di istruzioni per l'8086/8088.

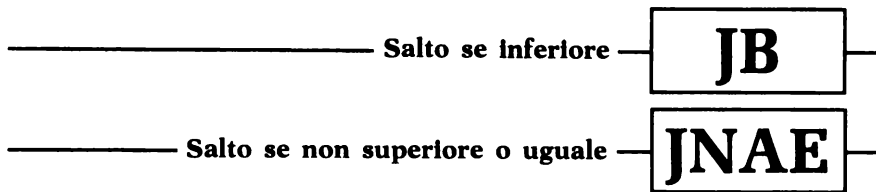
Esempio:

JAE label
JNB label

Codifica:

01110011	spostamento

Salto se inferiore — JB

Salto se non superiore o uguale — JNAE

Nomi mnemonici:

JB spostamento ad 8 bit con segno
JNAE spostamento ad 8 bit con segno

Funzione:

Se CF=1, allora IP=IP + spostamento (esteso a 16 bit dal segno).

Flag influenzati:

Nessuno

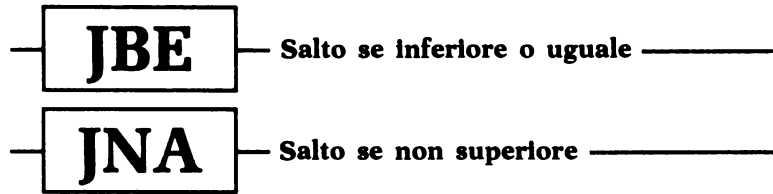
Descrizione:

JB e JNAE trasferiscono il controllo all'operando "obiettivo" (IP + spostamento) se la condizione (CF=1) è inferiore / non superiore o uguale al valore testato. Entrambi questi nomi mnemonici genereranno la stessa istruzione per l'8086/8088.

Esempio:

```
JB   label
JNAB label
```

Codifica:

01110010	spostamento

Nomi mnemonici:

JBE label ad 8 bit con segno
JNA label ad 8 bit con segno

Funzione:

Se CF o ZF=1, allora IP=IP + label a 8 bit con segno (estesa a 16 bit dal segno)

Flag influenzati:

Nessuno

Descrizione:

JBE e JNA trasferiscono il controllo dell'operando finale (IP + spostamento) se le condizioni (CF o ZF=1) sono inferiori o uguali / o non superiori al valore testato. Questi due nomi mnemonici genereranno la stessa istruzione per l'8086/8088.

Esempio:

JBE label
JNA label

Codifica:

01110110	spostamento

Nome mnemonico:

JC label ad 8 bit con segno

Funzione:

Se CF=1, allora IP=IP + label a 8 bit con segno (estesa a 16 bit dal segno).

Flag influenzati:

Nessuno

Descrizione:

JC trasferisce il controllo all'operando "obiettivo" (IP spostamento) se CF=1.

Esempio:

JC label

Codifica:

01110010	spostamento

JCXZ — Salto se il registro CX è zero

Nome mnemonico:

JCXZ label ad 8 bit con segno

Funzione:

Se CX=0, allora IP=IP + spostamento (esteso a 16 bits dal segno).

Flag influenzati:

Nessuno

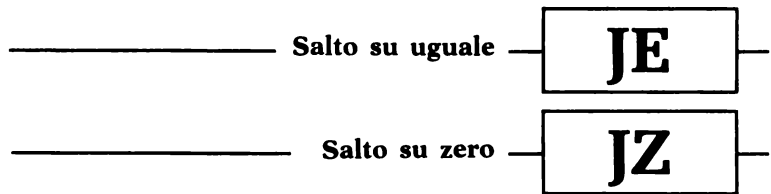
Descrizione:

JCXZ tasferisce il controllo all'operando obiettivo se CX è 0. Quest'istruzione è utile all'inizio di un loop per scavalcare il loop se CX ha valore 0 (cioè per eseguire il ciclo 0 volte).

Esempio:

JCXZ label

Codifica:

11100011	spostamento

Nomi mnemonici:

JE label ad 8 bit con segno
JZ label ad 8 bit con segno

Funzione:

Se ZF=1, allora IP=IP + spostamento (esteso a 16 bit dal segno).

Flag influenzati:

Nessuno

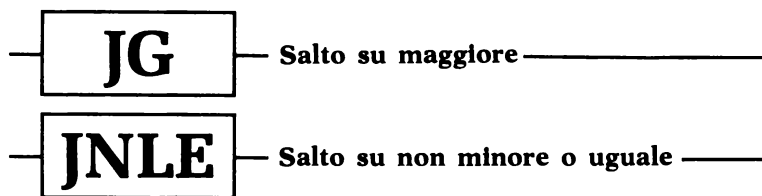
Descrizione:

JE e JZ trasferiscono il controllo all'operando obiettivo (IP + spostamento) se la condizione (ZF=1) è uguale/zero sul valore testato. Entrambi questi mnemonici generano la stessa istruzione per l'8086/8088.

Esempio:

JE label
JZ label

Codifica:

01110100	spostamento

Nomi mnemonici:

JG label ad 8 bit con segno
JNLE label ad 8 bit con segno

Funzione:

Se SF=OF or ZF=0, allora IP=IP + spostamento (esteso a 16 bit dal segno)

Flag influenzati:

Nessuno

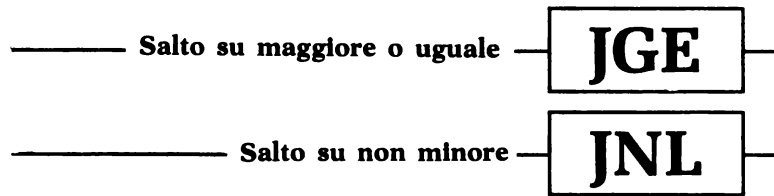
Descrizione:

JG e JNLE trasferiscono il controllo all'operando obiettivo (IP + spostamento) se la condizione SF xor OF=0 or ZF=0 è maggiore / non minore o uguale al valore testato. Entrambi questi nomi mnemonici genereranno la stessa istruzione per l'8086/8088.

Esempio:

JG label
JNLE label

Codifica:

01111111	spostamento

Nomi Mnemonici:

JGE spostamento ad 8 bit con segno
JNL spostamento ad 8 bit con segno

Funzione:

Se SF è uguale a OF, allora IP=IP + spostamento (esteso a 16 bit dal segno).

Flag Influenzati:

Nessuno.

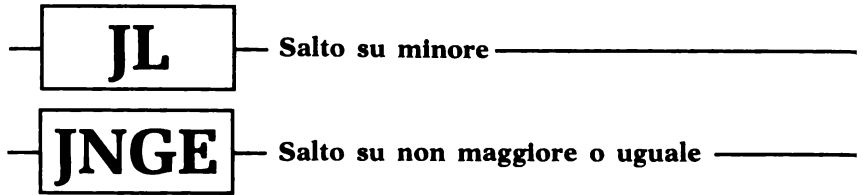
Descrizione:

JGE e JNL trasferiscono il controllo all'operando obiettivo (IP + spostamento) se la condizione (SF XOR OF=0) è maggiore o uguale / non minore al valore testato. Questi due nomi mnemonici genereranno la stessa istruzione per l'8086/8088.

Esempio:

JGE label
JNL label

Codifica:

01111101	Spostamento

Nomi mnemonici:

JL spostamento ad 8 bit con segno
JNGE spostamento ad 8 bit con segno

Funzione:

Se SF non è uguale a OF, allora IP=IP + spostamento (esteso a 16 bit dal segno).

Flag influenzati:

Nessuno

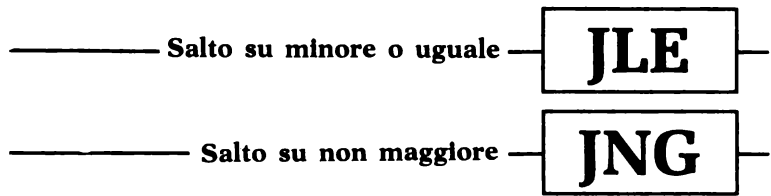
Descrizione:

JL e JNGE trasferiscono il controllo all'operando obiettivo (IP + spostamento) se la condizione (SF XOR OF=1) è minore / non maggiore o uguale al valore testato. Entrambi questi nomi mnemonici genereranno la stessa istruzione per l'8086/8088.

Esempio:

JL label
JNGE label

Codifica:

01111100	spostamento

Nomi mnemonici:

JLE label ad 8 bit con segno
JNG label ad 8 bit con segno

Funzione:

Se SF non è uguale a OF o ZF=1, allora IP=IP + spostamento (esteso a 16 bit dal segno)

Flag influenzati:

Nessuno

Descrizione:

JLE e JNG trasferiscono il controllo all'operando obiettivo (IP + spostamento) se la condizione SF XOR OF=1 o ZF=1 è minore o uguale / non maggiore del valore testato. Entrambi questi nomi mnemonici genereranno la stessa istruzione per l'8086/8088.

Esempio:

JLE label
JNG label

Codifica:

01111110	spostamento

JMP — Salto incondizionato

Nome mnemonico:

JMP obiettivo

Funzione:

Se intrasegmento, allora IP=IP + spostamento con segno;
Se intersegmento, allora CS e IP vengono sostituiti dai nuovi valori ottenuti dall'istruzione.

Flag influenzati:

Nessuno

Descrizione:

JMP trasferisce incondizionatamente il controllo alla locazione obiettivo. L'operando obiettivo può essere ottenuto dall'istruzione stessa (JMP diretto); o dalla memoria, o da un registro a cui ci si riferisce nell'istruzione (JMP indiretto).

Un JMP intrasegmento diretto cambia il puntatore all'istruzione aggiungendo lo spostamento relativo dell'obiettivo dall'istruzione JMP. Se l'Assembler può determinare che l'obiettivo si trova entro 127 byte dal JMP, esso genera automaticamente un'istruzione a due byte SHORT JMP. Altrimenti l'Assembler indirizzerà un obiettivo entro + o − 32K.

Un JMP intrasegmento può essere fatto usando la memoria o un registro generale a 16 bit. Nel primo caso il contenuto della parola a cui ci si riferisce nell'istruzione sostituisce IP. Nel secondo caso, il nuovo valore di IP viene preso dal registro nominato nell'istruzione.

Un JMP intersegmento sostituisce IP e CS con i valori contenuti nell'istruzione. Il JMP intersegmento indiretto può essere fatto solo attraverso la memoria: la prima parola del puntatore (lungo due parole) a cui si fa riferimento nell'istruzione sostituisce IP, la seconda parola sostituisce CS.

Esempio:

JMP label
JMP label.seg

Codifica:

Intrasegmento diretto

11101001	spost-basso	spost-alto

Intrasegmento diretto corto (SHORT JMP)

Lo spostamento è esteso a 16 bit dal segno

11101011	spostamento

Intrasegmento indiretto

11111111	mod 100 r/m

Intersegmento diretto

11101010	offset-basso	offset-alto	seg-basso	seg-alto

Intersegmento indiretto

11111111	mod 101 r/m

JNC — Salto se il riporto è 0

Nome mnemonico:

JNC label ad 8 bit con segno

Funzione:

Se CF=0, allora IP=IP + spostamento (esteso a 16 bit dal segno).

Flag influenzati:

Nessuno

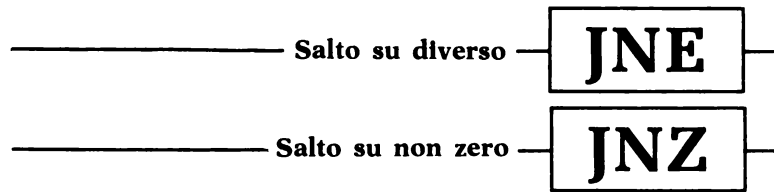
Descrizione:

JNC trasferisce il controllo all'operando obiettivo (IP + spostamento) a condizione che CF=0.

Esempio:

JNC label

Codifica:

01110011	spostamento

Nomi mnemonici:

JNE label ad 8 bit con segno
JNZ label ad 8 bit con segno

Funzione:

Se ZF=0, allora IP=IP + spostamento (esteso a 16 bit dal segno).

Flag influenzati:

Nessuno

Descrizione:

JNE e JNZ trasferiscono il controllo all'operando obiettivo (IP + spostamento) se la condizione testata (ZF=0) è vera.

Esempio:

JNE label
JNZ label

Codifica:

01110101	spostamento

JNO — Salto su non overflow

Nome mnemonico:

JNO spostamento ad 8 bit con segno

Funzione:

Se OF = 0, allora IP = IP + spostamento (esteso a 16 bit dal segno)

Flag influenzati:

Nessuno

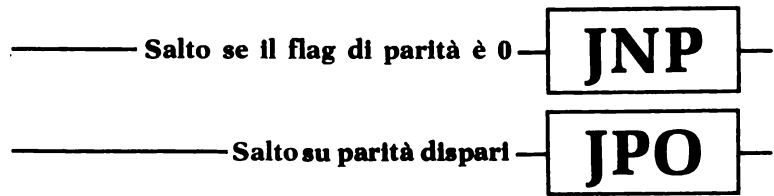
Descrizione:

JNO trasferisce il controllo all'operando obiettivo (IP + spostamento) se la condizione testata (OF = 0) è vera

Esempio:

JNO label

Codifica:

01110001	spostamento

Nomi mnemonici:

JNP spostamento ad 8 bit con segno
JNO spostamento ad 8 bit con segno

Funzione:

Se PF = 0, allora IP = IP + spostamento (esteso a 16 bit dal segno)

Flag influenzati:

Nessuno

Descrizione:

JNP e JPO trasferiscono il controllo all'operando obiettivo (IP + spostamento) se la condizione testata (PF = 0) è vera. Entrambi questi nomi mnemonici genereranno la stessa istruzione per l'8086/8088.

Esempio:

JNP label
JPO label

Codifica:

01111011	spostamento

JNS — Salto se il flag di segno è 0

Nome mnemonico:

JNS spostamento ad 8 bit

Funzione:

Se SF = 0, allora IP = IP + spostamento (esteso a 16 bit dal segno)

Flag influenzati:

Nessuno

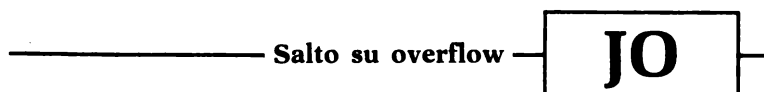
Descrizione:

JNS trasferisce il controllo all'operando obiettivo (IP + spostamento) se la condizione testata (SF = 0) è vera.

Esempio:

JNS label

Codifica:

01111001	spostamento

Nome mnemonico:

JO spostamento ad 8 bit con segno

Funzione:

Se OF = 1, allora IP = IP + spostamento (esteso a 16 bit dal segno).

Flag influenzati:

Nessuno

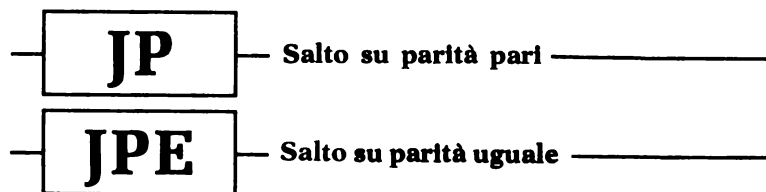
Descrizione:

JO trasferisce il controllo all'operando obiettivo (IP + spostamento) se la condizione testata (OF = 1) è vera.

Esempio:

JO label

Codifica:

01110000	spostamento

Nomi mnemonici:

JP spostamento ad 8 bit con segno
JPE spostamento ad 8 bit con segno

Funzione:

Se PF = 1, allora IP = IP + spostamento (esteso a 16 bit dal segno)

Flag influenzati:

Nessuno

Descrizione:

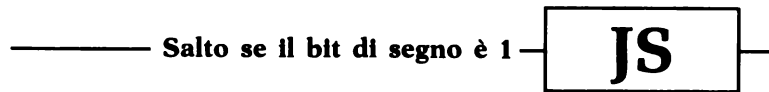
JP e JPE trasferiscono il controllo dell'operando obiettivo (IP + spostamento) se la condizione testata (PF = 1) è vera.

Esempio:

JP label
JPE label

Codifica:

01111010	spostamento

Salto se il bit di segno è 1 — JS

Nome mnemonico:

JS spostamento ad 8 bit con segno

Funzione:

Se SF = 1, allora IP = IP + spostamento (esteso a 16 bit dal segno)

Flag influenzati:

Nessuno

Descrizione:

JS trasferisce il controllo all'operando obiettivo (IP + spostamento) se la condizione testata (SF = 1) è vera.

Esempio:

JS label

Codifica:

01111000	spostamento

LAHF — Carica il registro AH dai flag

Nome mnemonico:

LAHF nessun operando

Funzione:

AH = d7 d6 d5 d4 d3 d2 d1 d0
SF ZF X AF X PF X CF dove X = non definito.

Flag influenzati:

Nessuno

Descrizione:

LAHF (carica il registro AH dai flag) copia rispettivamente SF, ZF, AF, PF, CF nei bits 7, 6, 5, 4, 2, 0 del registro AH. I contenuti dei bits 5, 3, e 1 sono non definiti.

Codifica:

10011111

Nome mnemonico:

LDS destinazione, sorgente

Funzione:

REG = dati dall'operando sorgente
DS = dati dall'operando sorgente + 2

Flag influenzati:

Nessuno

Descrizione:

LDS trasferisce una variabile puntatore di 32 bit dall'operando sorgente, che deve essere un operando in memoria, all'operando destinazione ed al registro DS. La parola offset puntatore viene trasferita nell'operando destinazione, che può essere un qualsiasi registro generale a 16 bit. La parola segmento del puntatore viene trasferita nel registro DS.

Esempio:

LDS SI, label

Codifica:

11000101	mod reg r/m

LEA — Carica un indirizzo effettivo

Nome mnemonico:

LEA destinazione, sorgente

Funzione:

REG = offset dell'operando sorgente

Flag influenzati:

Nessuno

Descrizione:

LEA trasferisce l'offset dell'operando sorgente (anzichè il suo valore) all'operando destinazione. L'operando sorgente deve essere un'operando in memoria e l'operando destinazione deve essere un registro generale a 16 bit.

Esempio:

LEA CX, label

Codifica:

10001101	mod reg r/m

Nome mnemonico:

LES destinazione, sorgente

Funzione:

REG = dati dall'indirizzo sorgente
ES = dati dall'indirizzo sorgente + 2

Flag influenzati:

Nessuno

Descrizione:

LES trasferisce una variabile puntatore di 32 bit dall'operando sorgente, che deve essere un operando in memoria, all'operando destinazione ed al registro ES. La parola offset del puntatore viene trasferita all'operando destinazione, che può essere un qualsiasi registro generale a 16 bit. La parola segmento del puntatore viene trasferita al registro ES.

Esempio:

LES DI, label	
Label	0215h
Label + 2	0457h
Prima:	*Dopo:*
DI = 0158h	DI = 0215h
ES = 1005h	ES = 0457h

Codifica:

11000100	mod reg r/m

Nome mnemonico:

LOCK MOV AX,Label

Funzione:

Nessuna

Flag influenzati:

Nessuno

Descrizione:

LOCK è un prefisso di 1 byte che fa sì che l'8088 (configurato nel modo massimo) mantenga attivo il segnale di LOCK bus mentre esegue l'istruzione successiva.

Codifica:

11110000

— Carica una stringa (byte o parola) — LODS

Nome mnemonico:

LODS stringa-sorgente

Funzione:

Se stringa di byte: AL = dati in memoria indirizzati da SI:

Se DF = 0, allora SI = SI + 1
Se DF = 1, allora SI = SI − 1

Se stringa di parole: AX = dati in memoria indirizzati da SI:

Se DF = 0, allora SI = SI + 2
Se DF = 1, allora SI = SI − 2

Flag influenzati:

Nessuno

Descrizione:

LODS trasferisce l'elemento stringa (byte o parola) indirizzato da SI al registro AL o AX, e aggiorna SI per puntare all'elemento successivo della stringa. L'assembler usa la stringa sorgente specificata nell'istruzione per determinare il tipo e l'accessibilità dei dati.

Codifica:

1010110w

LOOP — Loop se CX non è 0

Nome mnemonico:

LOOP spostamento ad 8 bit con segno

Funzione:

CX = CX − 1: se CX non è uguale a 0, allora IP = IP + spostamento (esteso a 16 bit dal segno). Se CX = 0, viene eseguita l'istruzione sequenziale successiva.

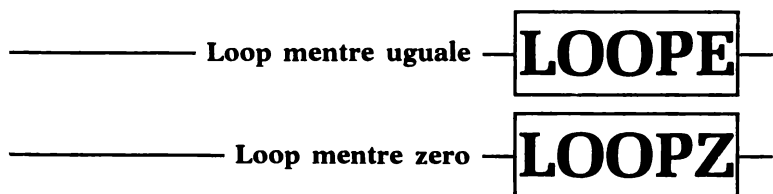
Flag influenzati:

Nessuno

Descrizione:

LOOP decrementa CX di 1 e trasferisce il controllo all'operando obiettivo se CX non è 0; altrimenti viene eseguita l'istruzione che segue LOOP.

Codifica:

11100010	spostamento

Nomi mnemonici:

LOOPE spostamento ad 8 bit con segno
LOOPZ spostamento ad 8 bit con segno

Funzione:

CX = CX − 1: se CX non è uguale a 0 e ZF = 1, allora
IP = IP + spostamento (esteso a 16 bit dal segno).

Flag influenzati:

Nessuno

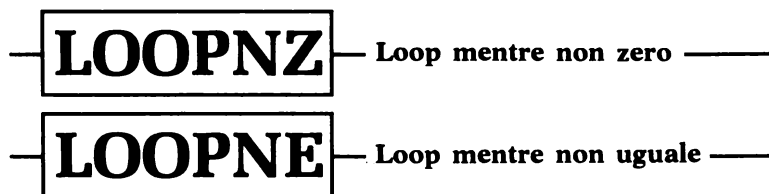
Descrizione:

LOOPE e LOOPZ sono nomi mnemonici differenti per la stessa istruzione. CX viene decrementato di 1, ed il controllo viene trasferito all'operando obiettivo se CX non è 0 e ZF = 1; altrimenti si esegue l'istruzione che segue LOOPE/LOOPZ.

Esempio:

LOOPE label
LOOPZ label

codifica:

11100001	spostamento

Nomi mnemonici:

LOOPNZ spostamento ad 8 bit con segno
LOOPNE spostamento ad 8 bit con segno

Funzione:

CX = CX −1: se CX non è uguale a 0 e ZF = 0, allora IP = IP + spostamento (esteso a 16 bit dal segno).

Flag influenzati:

Nessuno

Descrizione:

LOOPNZ e LOOPNE sono nomi mnemonici differenti per la stessa istruzione. CX viene decrementato di 1, e il controllo viene trasferito all'operando obiettivo se CX non è 0 e ZF = 0; altrimenti si esegue l'istruzione che segue LOOPNE/LOOPNZ.

Esempio:

LOOPNZ label
LOOPNE label

Codifica:

11100000	spostamento

Trasferimento di un byte o di una parola

Nome mnemonico:

MOV destinazione, sorgente

Funzione:

Destinazione = sorgente

Flag influenzati:

Nessuno

Descrizione:

MOV trasferisce un byte o una parola dall'operando sorgente all'operando destinazione

Esempio:

```
MOV AX,BX
MOV CL,AL
MOV BX,300
MOV BH,25
```

Codifica:

Operando in un registro o in memoria a/da operando in un registro

100010dw	mod reg r/m

Da operando immediato a operando in un registro o in memoria

1100011w	mod 000 r/m	dato	dato se w=1

Da operando immediato a registro

1011wreg	dato	dato se w = 1

Da operando memoria ad accumulatore

1010000w	indirizzo-basso	indirizzo-alto

Da accumulatore ad operando in memoria

1010001w	indirizzo-basso	indirizzo-alto

Da operando in un registro o in memoria a registro segmento

10001110	mod 0 reg r/m

Da registro segmento a operando in un registro o in memoria

10001100	mod 0 reg r/m

Nome mnemonico:

MOVS stringa-destinazione, stringa-sorgente

Funzione:

Stringa destinazione = stringa sorgente

Flag influenzati:

Nessuno

Descrizione:

MOVS trasferisce un byte o una parola dalla stringa sorgente (indirizzata da SI) alla stringa destinazione (indirizzata da DI) ed aggiorna SI e DI per puntare all'elemento successivo della stringa. Quando viene usata con REP, MOVS attua un trasferimento in blocco da memoria a memoria. Gli operandi della stringa destinazione e della stringa sorgente specificati nell'istruzione vengono usati dall'assembler per determinare tipo ed accessibilità degli operandi.

Esempio:

MOVS Buffer1, Buffer2

Codifica:

1010010w

MUL — Moltiplicazione

Nome mnemonico:

MUL sorgente

Funzione:

Se è una sorgente byte, allora AX = AL × sorgente (senza segno)
Se è una sorgente parola, allora AX, DX = AX × sorgente (senza segno)

Flag definiti:

CF, OF

Flag indefiniti:

AF, PF, SF, ZF

Descrizione:

MUL esegue una moltiplicazione senza segno dell'operando sorgente per l'accumulatore. Se la sorgente è un byte essa viene moltiplicata per il registro AL, e il risultato a doppia lunghezza viene salvato in AH e AL. Se l'operando sorgente è uuna parola essa viene moltiplicata per il registro AX, e il risultato a doppia lunghezza viene salvato nei registri DX e AX. Gli operandi vengono trattati come numeri binari senza segno. Se la metà superiore del risultato (AH per la sorgente byte, DX per la sorgente parola) è non-zero CF e OF vengono posti a 1; altrimenti essi vengono azzerati. Quando CF e OF valgono 1, indicano che AH o DX contengono cifre significative del risultato:

Esempio:

```
MUL 25
MUL CX
MUL BL
```

Codifica:

1111011w	mod 100 r/m

Negazione (formazione del complemento a 2) — NEG

Nome mnemonico:

NEG destinazione

Funzione:

Destinazione = 0 − destinazione

Flag definiti:

AF, CF, OF, PF, SF, ZF

Descrizione:

NEG sottrae l'operando destinazione, che può essere un byte o una parola, da zero e ritorna il risultato alla destinazione. Cioè forma il complemento a 2 del numero.

Codifica:

1111011w	mod 011 r/m

NOP — Nessuna operazione

Nome mnemonico:

NOP

Funzione:

Nessuna

Flag influenzati:

Nessuno

Descrizione:

NOP non fa compiere alcuna azione alla CPU.

Codifica:

10010000

Negazione logica (formazione del complemento) — NOT

Nome mnemonico:

NOT destinazione

Funzione:

Destinazione = complemento a 1 della destinazione

Flag influenzati:

Nessuno

Descrizione:

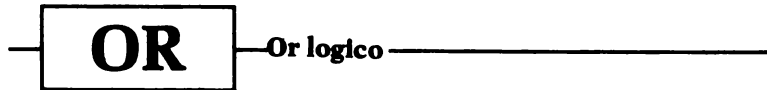
NOT inverte i bit (forma il complemento a 1) dell'operando byte o parola.

Esempio:

NOT AL

Codifica:

1111011w	mod 010 r/m

OR — Or logico

Nome mnemonico:

OR destinazione, sorgente

Funzione:

Destinazione = destinazione + OR sorgente: CF = 0:OF = 0

Flag definiti:

CF, OF, PF, SF, ZF

Flag indefiniti:

AF

Descrizione:

OR esegue l'"or inclusivo" logico dei due operandi (byte o parola) e memorizza il risultato nell'operando destinazione. Un bit nel risultato viene posto a 1 se uno dei due o entrambi i bit corrispondenti negli operandi originali valgono 1; altrimenti il bit nel risultato viene azzerato.

Esempio:

OR AL,BL
OR AL,00111010B

Codifica:

OR tra un operando in un registro o in memoria e un operando in un registro

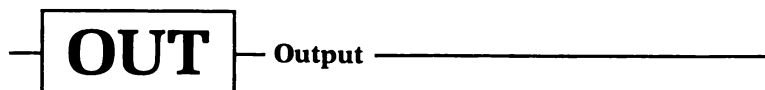
000001dw	mod reg r/m

OR tra un operando immediato e un operando destinazione in un registro o in memoria

100000sw	mod 001 r/m	dato	dato se w = 1

OR tra un operando immediato e l'accumulatore

0000110w	dato	dato se w = 1

OUT — Output

Nome mnemonico:

OUT porta, accumulatore

Funzione:

Se è specificato AL, allora i dati di AL vengono scritti al numero della porta
Se è specificato AX, allora AX viene scritto al numero della porta e al numero della porta + 1

Flag influenzati:

Nessuno

Descrizione:

OUT trasferisce un byte o una parola dal registro AL o AX ad una porta di output. Il numero della porta può essere specificato o con una costante immediata di un byte, e ciò permette l'accesso alle porte 0-255, o con un numero posto in precedenza nel registro DX, e ciò permette un accesso variabile alle porte 0-65535.

Esempio:

```
OUT 45,AX
OUT DX,AL
```

Codifica:

Porta fissa

1110011w	porta

Porta variabile: DX è l'indirizzo della porta

1110111w	operando immediato ad accumulatore

Nome mnemonico:

POP destinazione

Funzione:

Destinazione = dati al top dello stack: SP = SP + 2

Flag influenzati:

Nessuno

Descrizione:

POP trasferisce la parola che si trova all'attuale top dello stack (a cui punta SP) all'operando destinazione, poi incrementa SP di 2, puntando al nuovo top dello stack.

Esempio:

POP DX
POP DS (CS non si può specificare in POP)

Codifica:

Operando in memoria o in un registro

10001111	mod 000 r/m

Operando in un registro

01011reg

Registro segmento

000reg111

POPF Prelevamento dallo stack dei valori dei flag

Nome mnemonico:

POPF

Funzione:

Ai bit dei flag vengono assegnati i valori prelevati dal top dello stack: SP = SP + 2

Flag influenzati:

Tutti

Descrizione:

POPF trasferice i bit specifici dalla porta al top.dello stack (a cui punta il registro SP) ai flag, rimpiazzando perciò qualsiasi valore fosse precedentemente contenuto nei flag. SP viene poi incrementato di 2.

POPF può anche essere usato per assegnare un valore a TF, dato che non vi è alcuna istruzione specifica per fare ciò.

Codifica:

10011100

Deposito di un dato al top dello stack

Nome mnemonico:

PUSH sorgente

Funzione:

SP = SP – 2: l'operando sorgente viene memorizzato al top dello stack

Flag influenzati:

Nessuno

Descrizione:

PUSH decrementa SP (il puntatore allo stack) di 2 e quindi trasferisce una parola dall'operando sorgente al top dello stack a cui ora punta SP.

Esempio:

PUSH SI
PUSH ES (CS è legale in PUSH)

Codifica:

Operando in un registro o in memoria

Operando in un registro

01010reg

Registro segmento

000reg110

PUSHF — Push dei flag

Nome mnemonico:

PUSHF

Funzione:

SP = − 2: il contenuto del registro dei flag viene memorizzato al top dello stack

Flag influenzati:

Nessuno

Descrizione:

PUSHF decrementa SP (il puntatore allo stack) di 2 e quindi trasferisce tutti i flag alla parola al top dello stack a cui punta SP. I flag non vengono influenzati.

Codifica:

10011101

Rotazione a sinistra attraverso il riporto

Nome mnemonico:

RCL destinazione, contatore

Funzione:

(temp) = contatore: se temp non è = 0, allora (riportemp) = CF; CF = bit di ordine alto della destinazione; destinazione = (destinazione x 2) + (riportemp); (temp) = temp − 1; ripetere la funzione fino a che temp = 0. Se contatore = 1, allora: se il bit di ordine alto della destinazione non è uguale a CF, allora OF = 1; altrimenti OF = 0.
Se contatore non è uguale a 1, allora OF è indefinito.

Flag definiti:

CF, OF

Descrizione:

RCL ruota i bit dell'operando destinazione (byte o parola) a sinistra per il numero di bit specificati nell'operando contatore. Il flag del riporto (CF) viene trattato come "parte dell'" operando destinazione; cioè il suo valore viene ruotato nel bit di ordine basso della destinazione e viene sostituito dal bit di ordine alto della destinazione.

Esempio:

RCL AL,3

Prima:	*Dopo:*
AL = 01011110, CF = 0	AL = 10111100, CF = 0 prima rotazione
AL = 10111100, CF = 0	AL = 01111000, CF = 1 seconda rotazione
AL = 01111000, CF = 1	AL = 11110001, CF = 0 terza rotazione

Codifica:

110100vw	mod 010 r/m

Se v = 0, allora contatore = 1

RCR — Rotazione a destra attraverso il riporto

Nome mnemonico:

RCR destinazione, contatore

Funzione:

(temp) = contatore: se temp non è uguale a 0, allora: (riportemp) = CF: CF = bit di ordine basso della destinazione;
Destinazione = destinazione / 2;
bit di ordine alto della destinazione = (riportemp); (temp) = (temp) − 1.
Se contatore = 1, allora: se il bit di ordine alto della destinazione non è uguale al successivo bit di ordine alto della destinazione, allora OF = 1; altrimenti, OF = 0.
Se contatore non è uguale a 1, allora OF è indefinito.

Flag influenzati:

CF, OF

Descrizione:

RCR ruota i bit dell'operando destinazione (byte o parola) a destra per il numero di bit specificato nell'operando contatore. Il flag del riporto (CF) viene trattato come "parte dell'" operando destinazione; cioè il suo valore viene ruotato nel bit di ordine alto della destinazione e viene sostituito dal bit di ordine basso della destinazione.

Esempio:

RCR BL,2

Prima:	*Dopo*
BL = 11000010, CF = 1	BL = 11100001, CF = 0 Prima rotazione
BL = 11100001, CF = 0	BL = 01110000, CF = 1 Seconda rotazione

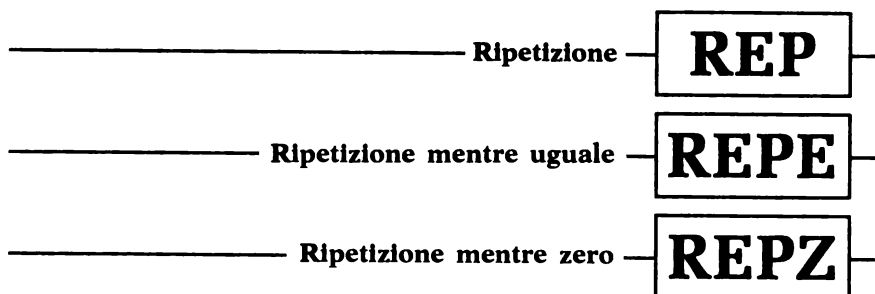
Codifica:

110100vw	mod 011 r/m

Se v = 0, allora contatore =1

Ripetizione — REP

Ripetizione mentre uguale — REPE

Ripetizione mentre zero — REPZ

Nome mnemonico:

REP MOVS destinazione, sorgente

Flag influenzati:

Nessuno

Descrizione:

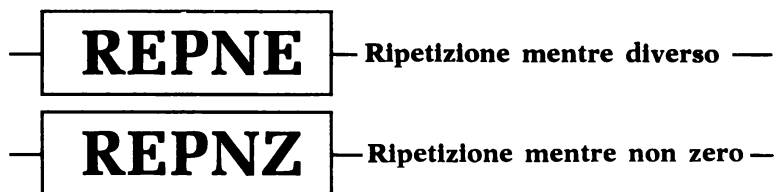
REP viene usata insieme alle istruzioni MOVS (trasferimento di una stringa) e STOS (memorizzazione di una stringa) e viene interpretata come "ripetere mentre non si è alla fine della stringa" (CX non 0).
REPE e REPZ funzionano allo stesso modo e fisicamente sono le stesse byte prefisso di REP. Queste istruzioni vengono usate con le istruzioni CMPS (confronto fra stringhe) e SCAS (scansione di una stringa) e richiedono che ZF (impiegato da queste istruzioni) venga posto a 1 prima di iniziare la ripetizione successiva. ZF non deve essere necessariamente inizializzato prima di iniziare l'esecuzione dell'istruzione da ripetere sulla stringa.

Le sequenze di ripetizione sulle stringhe si possono interrompere; il processore riconoscerà l'interruzione prima di elaborare l'elemento successivo della stringa. Il processo di elaborazione delle interruzioni di sistema non viene influenzato in nessun modo. Dopo il ritorno dall'interruzione l'operazione oggetto della ripetizione viene ripresa dal punto in cui era stata interrotta.

Esempio:

```
REP  MOVS destinazione, sorgente
REPE CMPS destinazione, sorgente
```

Codifica:

1111001z

Nome mnemonico:

REPNE SCAS destinazione:

Flag influenzati:

Nessuno

Descrizione:

REPNE e REPNZ funzionano allo stesso modo e sono fisicamente lo stesso byte prefisso di REP. Queste istruzioni vengono usate con le istruzioni CMPS (confronto fra stringhe) e SCAS (scansione di una stringa), ZF deve essere azzerato o la ripetizione viene terminata, ZF non deve essere necessariamente inizializzato prima di iniziare l'esecuzione dell'istruzione da ripetere sulla stringa.

Le sequenze di ripetizione sulla stringa possono essere interrotte; il processore riconoscerà l'interruzione prima di elaborare l'elemento successivo della stringa. Il processo di elaborazione delle interruzioni di sistema non viene influenzato in alcun modo. Dopo il ritorno dall'interruzione l'operazione oggetto della ripetizione viene ripresa dal punto in cui era stata interrotta.

Esempio:

REPNE SCAS destinazione

Codifica:

1111001z

Ritorno da una parocedura — RET

Nome mnemonico:

RET Valore-opzionale

Funzione:

Se intra-segmento: IP viene prelevato dallo stack;
SP = SP + 2
Se inter-segmento: CS viene prelevato dallo stack;
SP = SP + 2
IP viene prelevato dallo stack; SP = SP + 2
Se viene usato il valore opzionale, allora SP = SP + valore

Flag influenzati:

Nessuno

Descrizione:

RET ritrasferisce il controllo da una procedura all'istruzione che segue la CALL che ha attivato la procedura. L'assembler genera un RET intra-segmento se il programmatore ha definito la procedura NEAR (vicina), o un RET inter-segmento se la procedura è definita FAR (lontana). RET trasferisce la parola al top dello stack (a cui punta SP) nel puntatore all'istruzione (IP) e incrementa SP di 2. Se RET è inter-segmento, la parola che è ora al top dello stack viene prelevata e trasferita nel registro CS, ed SP viene incrementato di 2. Se è stato specificato un valore opzionale, RET aggiunge questo valore a SP. Questa caratteristica può essere usata per scartare i parametri messi nello stack prima dell'esecuzione dell'istruzione CALL.

Esempio:

RET
RET 6

Codifica:

Intra-segmento

11000011

Intra-segmento e addizione immediata al puntatore allo stack

11000011	dato-basso	dato-alto

Inter-segmento

11001011

Inter-segmento e addizione immediata al puntatore allo stack

11001010	dato-basso	dato-alto

Nome mnemonico:

ROL destinazione, contatore

Funzione:

(temp) = contatore; se (temp) non è uguale a 0, allora:
CF = bit di ordine alto della destinazione;
destinazione = destinazione × 2 + CF; (temp) = (temp) − 1
Se contatore = 1, allora: se il bit di ordine alto della destinazione non è uguale a CF, allora OF = 1; altrimenti OF = 0.
Se contatore non è uguale a 1 allora OF è indefinito.

Flag definiti:

CF, OF

Descrizione:

ROL ruota i bit nell'operando destinazione (byte o parola) a sinistra per il numero di bit specificati nell'operando contatore. Se il valore di contatore è uguale a 1 può essere specificato direttamente. Se il valore di contatore è maggiore di 1 deve essere posto nel registro CL prima di usare questa istruzione.

Esempio:

```
ROL BL,1
MOV CL,2

ROL BL,CL
```

Prima:	*Dopo:*
BL = 11001100, CF = 0	*BL* = 10011001,CF = 1 prima rotazione
BL = 10011001, CF = 1	BL = 00110011,CF = 1 seconda rotazione

Codifica:

110100vw	mod 000 r/m

Se v = 0, allora contatore = 1

ROR — Rotazione a destra

Nome mnemonico:

ROR destinazione, contatore

Funzione:

(temp) = contatore; se (temp) non è uguale a 0, allora:
CF = bit di ordine basso della destinazione;
destinazione = destinazione / 2
bit di ordine alto della destinazione = CF
(temp) = (temp) − 1
Se contatore = 1, allora: se il bit di ordine alto della destinazione non è uguale al successivo bit di ordine alto, allora OF =1; altrimenti, OF = 0.
Se contatore non è uguale a 1, allora OF è indefinito.

Flag influenzati:

CF, OF

Descrizione:

ROR ruota i bit nell'operando destinazione (byte o parola) a destra per il numero di bit specificato nell'operando contatore. Se l'operando contatore è uguale a 1 può essere specificato direttamente. Se l'operando contatore è maggiore di 1, deve essere posto nel registro CL prima di usare questa istruzione.

Esempio:

```
ROR BL,1
MOV CL,2
ROR BL,CL
```

Prima:	*Dopo:*
BL = 11001100, CF = 0	BL = 01100110, CF = 0 prima rotazione
BL = 01100110, CF = 0	BL = 00110011, CF = 0 seconda rotazione

Codifica:

110100vw	mod 001 r/m

Se v = 0, allora contatore = 1

– Immagazzina il registro AH nei flag – SAHF

Nome mnemonico:

SAHF

Funzione:

SF:	ZF:	X:	AF:	X:	PF:	X:	CF:	
d7	d6	d5	d4	d3	d2	d1	d0	AH trasferito nei flag

Flag definiti:

AF, CF, PF, SF, ZF

Descrizione:

SAHF trasferisce i bit 7, 6, 4, 2, e 0 dal registro AH rispettivamente in SF, ZF, AF, PF, e CF; perciò sostituisce i valori che questi flag avevano in precedenza. OF, DF, IF e TF non vengono influenzati.

Codifica:

10011110

Shift aritmetico a destra

Nome mnemonico:

SAR destinazione, contatore

Funzione:

(temp) = contatore; se (temp) non è uguale a zero, allora:
CF = bit di ordine basso della destinazione
destinazione = destinazione / 2
Il bit di ordine alto della destinazione è lo stesso del bit di ordine alto precedente (bit del segno); (temp) = (temp) − 1
Se contatore = 1, allora: se il bit di ordine alto della destinazione non è uguale al bit di ordine alto dopo il successivo, allora OF = 1; altrimenti OF = 0.

Flag definiti:

CF, OF, PF, SF, ZF

Flag indefiniti:

AF

Descrizione:

SAR sposta i bit dell'operando destinazione (byte o parola) a destra per il numero di bit specificato nell'operando contatore. Un bit uguale al bit di ordine alto originale (segno) viene fatto "entrare" a sinistra ad ogni spostamento e perciò si mantiene il segno del valore originale. È da notare che SAR non produce lo stesso risultato del dividendo di una istruzione IDIV "equivalente" se l'operando destinazione è negativo e i bits 1 vengono spostati all'esterno.
Per esempio lo spostare − 5 a destra di un bit, dà −3, mentre la divisione −5 / 2 dà −2.

La differenza fra le istruzioni è che IDIV tronca tutti i numeri verso lo zero, mentre SAR tronca i numeri positivi verso lo zero e quelli negativi verso l'infinito negativo.

Se contatore è uguale a 1 può essere specificato direttamente. Se contatore è maggiore di 1, il suo valore deve essere caricato nel registro CL prima di usare questa istruzione.

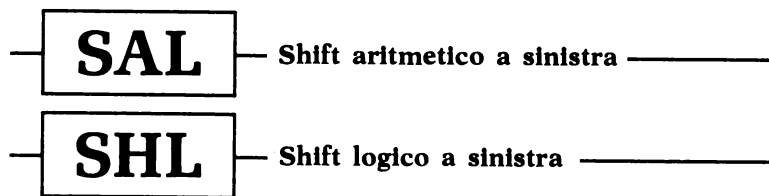
Esempio:

```
SAR BL,1
MOV CL,2
SAR BL,CL
```

Prima:	*Dopo:*
BL = 11001100, CF = 0	BL = 11100110, CF = 0 primo shift
BL = 11100110, CF = 0	BL = 11110011, CF = 0 secondo shift

Codifica:

110100vw	mod 111 r/m

Se v = 0, allora contatore = 1

Nomi mnemonici:

SAL destinazione, contatore
SHL destinazione, contatore

Funzione:

(temp) = contatore; se (temp) non è uguale a zero allora:
CF = bit di ordine alto della destinazione:
destinazione = destinazione × 2
bit di ordine basso della destinazione = 0;
(temp) = (temp) − 1
Se contatore = 1 allora: se il bit di ordine alto della destinazione non è uguale al bit CF allora OF = 1; altrimenti OF = 0.
Se contatore non è uguale a 1 allora OF è indefinito.

Flag definiti:

CF, OF, PF, SF, ZF

Flag indefiniti:

AF

Desrizione:

SHL e SAL eseguono la stessa operazione e sono fisicamente la stessa istruzione. Il byte, o la parola, di destinazione viene spostato a sinistra per il numero di bit specificati nell'operando contatore. Ad ogni spostamento viene fatto "entrare" uno 0 nel bit più a destra. Se il bit del segno mantiene il proprio valore originale, OF viene azzerato. Se il contatore è uguale a 1 può essere specificato direttamente. Se il contatore è maggiore di 1, il suo valore deve essere caricato nel registro CL prima di usare questa istruzione.

Esempio:

```
SAL AL,1
MOV CL,2
SAL AL,CL
```

Prima:	*Dopo:*
BL = 11001100, CF = 0	BL = 10011000, CF = 1 primo shift
BL = 10011000, CF = 1	BL = 00110000, CF = 1 secondo shift

Codifica:

110100vw	mod 100 r/m

Se v = 0, allora contatore = 1

SBB — Sottrazione con prestito

Nome mnemonico:

SBB destinazione, sorgente

Funzione:

Se CF = 1, allora destinazione = destinazione – sorgente – 1
Se CF diverso da 1, allora destinazione = destinazione – sorgente

Flag definiti:

AF, CF, OF, PF, SF, ZF

Descrizione:

SBB sottrae la sorgente dalla destinazione; quindi sottrae 1 se CF vale 1, e memorizza il risultato nell'operando destinazione. Entrambi gli operandi possono essere byte o parole. Entrambi gli operandi possono essere numeri binari con o senza segno.

Codifica:

Operando in memoria o in un registro e operando in un registro

000110dw	mod reg r/m

Operando sorgente immediato e operando destinazione in memoria o in un registro

100000sw	mod 011 r/m	dato	dato se s:w = 01

Operando sorgente immediato e operando destinazione nell'accumulatore

0001110w dato	dato se w = 1

Scansione di una stringa (byte o parola)

Nome mnemonico:

SCAS stringa-destinazione

Funzione:

Se la stringa destinazione è in byte, allora i dati indirizzati dal registro DI vengono sottratti da AL.
Se DF = 0, allora DI = DI + 1; Se DF = 1, allora DI = DI − 1.
Se la stringa destinazione è in parole, allora i dati indirizzati dal registro DI vengono sottratti da AX. Se DF = 0, allora DI = DI + 2; Se DF = 1, allora DI = DI − 2

Flag definiti:

AF, CF, OF, PF, SF, ZF

Descrizione:

SCAS sottrae l'elemento della stringa destinazione (byte o parola) indirizzato da DI dal contenuto di AL (stringa byte) o AX (stringa parola) e aggiorna i flag, ma non altera la stringa destinazione o l'accumulatore. SCAS aggiorna anche DI per puntare all'elemento successivo della stringa e aggiorna AF, CF, OF, PF, SF e ZF per riflettere la relazione del valore scandito AL / AX con l'elemento della stringa. SCAS può avere come prefisso REPE, REPZ, REPNE o REPNZ.

Codifica:

1010111w

SHR — Shift logico a destra

Nome mnemonico:

SHR destinazione, contatore

Funzione:

(temp) = contatore; se (temp) non è uguale a zero, allora CF = bit di ordine basso della destinazione = 0; (temp) = (temp) − 1.
Se contatore = 1, allora: se il bit di ordine alto della destinazione e' diverso dal successivo valore assunto dopo lo spostamento a destra allora OF = 1; altrimenti OF = 0
Se contatore non è uguale a 1, allora OF è indefinito.

Flag definiti:

CF, OF, PF, SF, ZF

Flag indefiniti:

AF

Descrizione:

SHR sposta i bit dell'operando destinazione (byte o parola) a destra per il numero di bit specificati nell'operando contatore. Ad ogni spostamento viene fatto entrare uno 0 nel bit più a sinistra. Se il bit del segno mantiene il valore originale, OF viene azzerato.

Se il contatore è uguale a 1 può essere specificato direttamente. Se il contatore è maggiore di 1, il suo valore deve essere caricato nel registro CL prima di eseguire questa istruzione.

Esempio:

```
SHR BL,1
MOV CL,2
SHR BL,CL
```

Prima:	*Dopo:*
BL = 00110011, CF = 0	BL = 00011001, CF = 1 primo shift
BL = 00011001, CF = 1	BL = 00001100, CF = 1 secondo shift

Codifica:

110100vw	mod 101 r/m

Se v = 0, allora contatore = 1

Nome mnemonico:

STC

Funzione:

CF = 1

Flag definiti:

CF

Descrizione:

STC posiziona CF a 1 e non influenza altri flag.

Codifica:

11111001

Nome mnemonico:

STD

Funzione:

DF = 1

Flag definiti:

DF

Descrizione:

STD posiziona DF a 1 e causa l'autodecremento dei registri indice SI e/o DI da parte delle istruzioni che trattano stringhe. STD non influenza altri flag. Se DF non vale 1 le istruzioni che manipolano stringhe eseguiranno un autoincremento.

Codifica:

11111101

STI — Set del flag di abilitazione delle interruzioni

Nome mnemonico:

STI

Funzione:

IF = 1

Flag definiti:

IF

Descrizione:

STI posiziona IF a 1, e perciò abilita la CPU a riconoscere le richieste di interruzioni mascherabili che appaiono sulla linea di input INTR. Un'interruzione in attesa non averrà riconosciuta effettivamente fino a che verrà eseguita l'istruzione che segue STI. STI non influenza altri flag.

Codifica:

11111011

Memorizzazione di una stringa (byte o parola) — STOS

Nome mnemonico:

STOS stringa-destinazione

Funzione:

Se è un byte, allora i dati di AL vengono immagazzinati all'indirizzo a cui punta DI.
Se DF = 0, allora DI = DI + 1.
Se DF = 1, allora DI = DI − 1.
Se è una parola, allora i dati di AX vengono immagazzinati all'indirizzo a cui punta DI.
Se DF = 0, allora DI = DI + 2.
Se DF = 1, allora DI = DI − 2.

Flag influenzati:

Nessuno

Descrizone:

STOS trasferisce un byte o una parola dal registro AL o AX all'elemento della stringa indirizzato da DI, ed aggiorna DI per puntare alla locazione successiva della stringa.

Esempio:

```
STOS BYTE_DEST
STOS WORD_DEST
REP STOS BYTE_DEST1
```

Codifica:

1010101w

Nome mnemonico:

SUB destinazione, sorgente

Funzione:

Destinazione = destinazione − sorgente

Flag definiti:

AF, CF, OF, PF, SF, ZF

Descrizione:

L'operando sorgente viene sottratto dall'operando destinazione, ed il risultato sostituisce l'operando destinazione. Gli operandi possono essere byte o parole. Entrambi gli operandi possono essere numeri binari con o senza segno.

Esempio:

```
SUB BX,AX
SUB BL,10
SUB SI,4790
```

Codifica:

Sottrazione tra un operando in un registro o in memoria e un operando in un registro

001010dw	mod reg r/m

Sottrazione di un operando immediato da un operando in memoria o in un registro

100000sw	mod 101 r/m	dato	dato se s:w = 01

Sottrazione di un immediato dall'accumulatore

0010110w	dato	dato se w = 1

Nome mnemonico:

TEST destinazione, sorgente

Funzione:

Viene eseguito un "and" tra la Destinazione e la Sorgente. I flag vengono aggiornati: CF = 0; OF = 0.

Flag definiti:

CF, OF, PF, SF, ZF

Flag indefiniti:

AF

Descrizione:

TEST esegue l' "and" logico dei due operandi (byte o parole), e aggiorna i segnali ma non ritorna il risultato (cioè nessuno dei due operandi viene cambiato). Se un'istruzione di test viene seguita da un'istruzione JNZ (salto se non zero), il salto verrà fatto se c'è qualche coppia di bit corrispondenti (ad esempio il terzo bit della sorgente e il terzo bit della destinazione) in cui entrambi i bit valgono 1.

Esempio:

TEST AL,3FH
TEST BX,0557H

Codifica:

Tra un operando in un registro o in memoria e un operando in un registro

1000010w	mod reg r/m

Tra un operando immediato e un operando in un registro o in memoria

1111011w	mod 000 r/m	dato	dato se w = 1

Tra un operando immediato e l'accumulatore

1010100w	dato	dato se w = 1

Nome mnemonico:

WAIT

Funzione:

Nessuna

Flag influenzati:

Nessuno

Descrizione:

WAIT fa entrare la CPU in stato di attesa mentre la sua linea test non è attiva.

Codifica:

10011011

XCHG — Scambio

Nome mnemonico:

XCHG destinazione, sorgente

Funzione:

(temp) = destinazione; destinazione = sorgente; sorgente = (temp).

Flag influenzati:

Nessuno

Descrizione:

XCHG scambia i contenuti degli operandi sorgente e destinazione (byte o parola).

Esempio:

XCHG BX,AX
XCHG label,CX

Codifica:

Tra un operando in memoria o in un registro e un operando in un registro.

1000011w	mod reg r/m	

Tra un operando in un registro e l'accumulatore

10010reg

Nome mnemonico:

XLAT tabella_nome

Funzione:

AL = dato all'indirizzo a cui punta BX + AL.

Flag influenzati

Nessuno

Descrizione:

XLAT sostituisce un byte nel registro AL con un byte ricavato da una tabella di traduzione a 256 byte codificata dall'utente. Si assume che il registro BX punti all'inizio della tabella. Il byte di AL viene usato nella tabella come un indice e viene sostituito dal byte all'offset nella tabella corrispondente al valore binario di AL. Il primo byte della tabella ha un offset di zero.

Per esempio, se AL contiene 5H e il sesto elemento della tabella di traduzione contiene 33H, AL conterrà 33H dopo che l'istruzione XLAT è stata eseguita.

Esempio:

```
MOV BX,OFFSET_Tabella_Valore
XLAT Tabella_1
```

Codifica:

11010111

XOR — Or esclusivo

Nome mnemonico:

XOR destinazione, sorgente

Funzione:

Destinazione = destinazione xor sorgente; CF = 0; OF = 0.

Flag definiti:

CF, OF, PF, SF, ZF

Flag indefiniti:

AF

Descrizione:

XOR esegue l' "or esclusivo" logico dei due operandi e memorizza il risultato nell'operando destinazione. Un bit nel risultato viene posto a 1 se i bit corrispondenti degli operandi originali contengono valori opposti.

Esempio:

XOR AX,BX

Prima:	*Dopo:*
AX = 5857H	AX = 00FFH
BX = 58A8H	BX = 58A8H

Codifica:

XOR tra un operando in memoria o in un registro e un operando in un registro

001100dw	mod reg r/m

XOR tra un operando sorgente immediato e un operando destinazione in memoria o in un registro

1000000w	mod 110 r/m	dato	dato se w = 1

XOR tra un operando sorgente immediato ed un operando destinazione nell'accumulatore

0011010w	dato	dato se w = 1

CAPITOLO 5

TECNICHE BASILARI DI PROGRAMMAZIONE

INTRODUZIONE

In questo capitolo esamineremo alcune tecniche basilari per programmare l'8086/8088. Scriveremo semplici programmi aritmetici usando molte delle istruzioni discusse nel Capitolo 4.

In più discuteremo le subroutines e i ritorni dalle subroutines. Vedremo che a causa della larghezza del bus indirizzi dell'8086/8088, ci sono parecchi modi con cui si può chiamare una subroutine.

PROGRAMMI ARITMETICI

I programmi aritmetici in questo capitolo vi mostreranno come eseguire addizione, sottrazione, moltiplicazione e divisione con l'8086/8088. Questi programmi vi insegneranno ad eseguire operazioni aritmetiche su numeri binari positivi e su numeri negativi rappresentati come interi in complemento a 2.

Cominceremo con un esempio di addizione ad 8 bit (si noti, tuttavia, che l'8086/8088 è anche in grado di effettuare operazioni aritmetiche con numeri di 16 bit).

Addizione ad 8 bit

Il programma di addizione che segue somma tra loro 2 operandi ad 8 bit, OP1 e OP2, memorizzati rispettivamente negli indirizzi di memoria ADR1 e ADR2. La somma, immagazzinata all'indirizzo di memoria ADR3, è chiamata RES. La Figura 5.1 mostra un diagramma a blocchi che descrive come gli operandi e il risultato sono immagazzinati nella memoria. Ecco il programma:

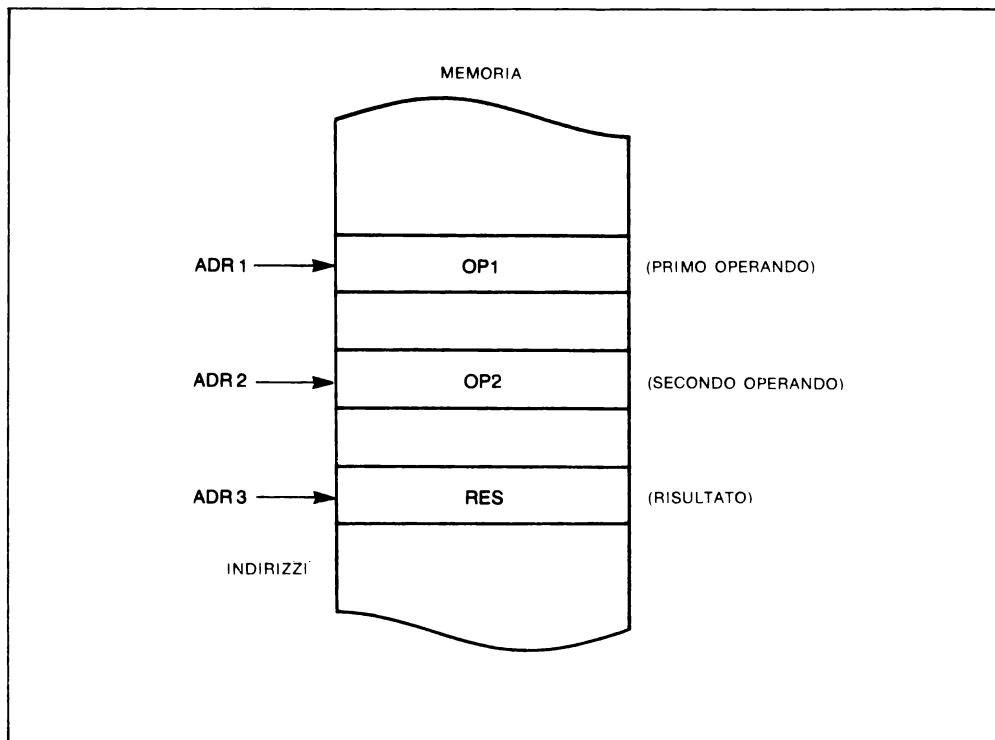



Figura 5.1 — Questo diagramma mostra come gli operandi ed il risultato sono immagazzinati nella memoria. OP1 è addizionato ad OP2 ed il risultato è immagazzinato nella locazione di memoria RES.

ISTRUZIONE	**COMMENTO**
MOV AL,ADR1	Trasferisce OP1 in AL
ADD AL,ADR2	Somma AL ad OP2 @ ADR2
MOV ADR3,AL	La somma viene immagazzinata in @ ADR3

Ogni linea del programma (qui espressa in forma simbolica) è chiamata istruzione. Ogni istruzione è tradotta dal calcolatore nel corrispondente codice oggetto che può essere eseguito dalla CPU. Più avanti esamineremo il codice oggetto per questo programma. Diamo ora un'occhiata al programma.

Descrizione del programma di addizione

La prima linea del programma specifica che nel registro AL (8 bit) devono essere immagazzinati i dati prelevati dalla locazione di memoria ADR1 (nota: al posto del registro AL avremmo potuto specificare uno qualsiasi dei registri generali a 8 bit nella CPU 8086/8088).

ADR1 è una rappresentazione simbolica dell'indirizzo reale

nella memoria del sistema. Questo indirizzo è definito altrove nel programma. Quando la CPU legge i dati dalla memoria il valore a 16 bit di ADR1 è sommato all'opportuno registro di segmentazione per generare l'indirizzo di sistema a 20 bit. Esaminiamo ora il campo COMMENTO, il più a destra di ogni linea di istruzioni. I commenti sono molto utili al programmatore per capire e ricordare esattamente cosa fa ciascuna linea di programma. I commenti sono ignorati quando viene generato il codice oggetto.

La Figura 5.2 mostra i risultati dopo che è stata eseguita la prima istruzione.

La seconda istruzione:

```
ADD AL,ADR2
```

specifica che il dato memorizzato all'indirizzo di memoria ADR2 deve essere sommato al contenuto (8 bit) del registro AL. L'indirizzo ADR2 contiene il secondo operando, OP2. Quando viene eseguita la seconda istruzione OP2 viene letto dalla memoria e quindi sommato ad OP1, poi il risultato viene immagazzinato in AL.

La somma di OP1 e OP2 è ora contenuta nel registro AL.

Per completare il programma, dobbiamo trasferire il contenuto di AL nella locazione di memoria ADR3. Questo compito è eseguito dalla terza linea del programma:

```
MOV ADR3,AL.
```

Esaminando questo semplice programma vediamo che ci sono alcuni punti importanti del modo di operare dell'8086/8088 che dovremmo discutere. Uno di tali punti è che i dati sono memorizzati in byte di memoria. Perciò, in questo esempio, quando il risultato è stato scritto nella memoria all'indirizzo ADR3, solo il byte ADR3 è stato disturbato. Questo è vero anche con il bus dati a 16 bit dell'8086. La CPU scriverà elettricamente un unico byte nella memoria del sistema perchè nell'istruzione viene usato un registro ad 8 bit (AL).

Scrittura di un byte in memoria

Un altro punto interessante del programma è che i dati vengono trasferiti dalla memoria a un registro interno. La locazione da cui sono letti i dati è chiamata operando sorgente mentre quella in cui sono scritti è chiamata operando destinazione. L'operando sorgente non viene cambiato durante l'esecuzione del programma. Se dovessimo esaminare le locazioni di memoria ADR1 e ADR2 alla fine del programma troveremmo che contengono gli stessi dati che erano originariamente presenti all'inizio del programma.

Operando sorgente e operando destinazione

Assegnamo ora dei valori numerici alle locazioni di indiriz-

La pseudo-istruzione EQU

zo ADR1, ADR2 e ADR3. Possiamo fare ciò usando pseudo istruzioni. Le pseudo istruzioni sono istruzioni che non vengono usate dal microprocessore, ma sono invece utilizzate dal programma del calcolatore che genera il codice oggetto. La pseudo istruzione che useremo qui è l'istruzione EQUATE, scritta come EQU. Il nome è derivato dalla funzione: l'istruzione eguaglia, o pone allo stesso valore, i due argomenti su ciascun lato della pseudo istruzione EQU. Un programma completo per questa addizione ad 8 bit è mostrato in Figura 5.3. In questo programma ADR1 è uguale a 300H, ADR2 è uguale a 320H e ADR3 è uguale a 345H.

Addizione a 16 bit

Estendiamo ora il problema appena affrontato e passiamo dall'addizione ad 8 bit all'addizione a 16 bit. Il problema a 16 bit può essere risolto esattamente come quello a 8 bit tranne che per una differenza nei registri usati e nel numero di locazioni di memoria considerate.

Si ricordi che con l'8086/8088 un byte è una singola locazione di memoria.

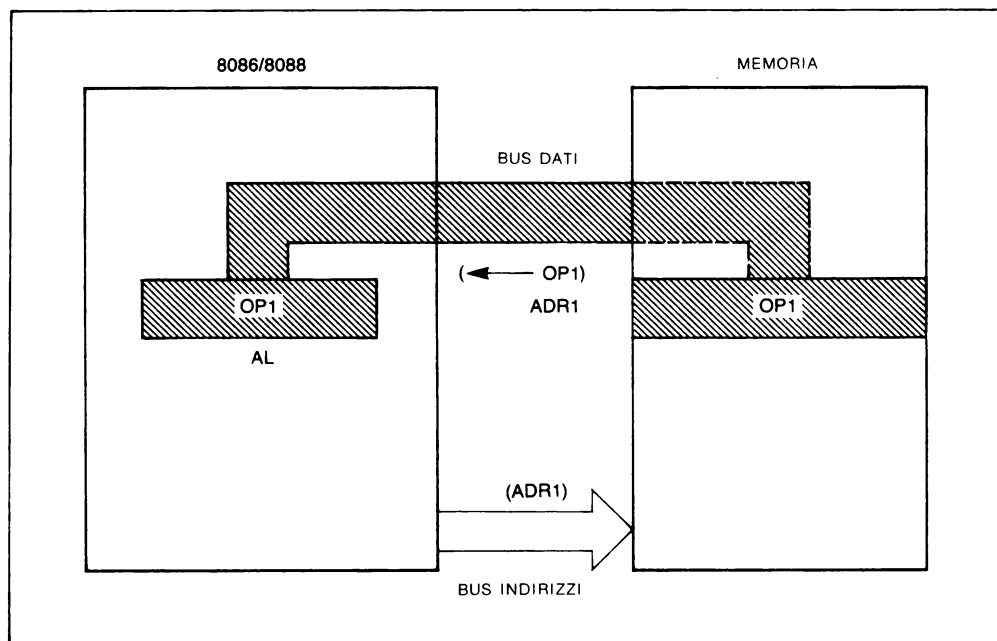



Figura 5.2 — Dopo che è stata eseguita l'istruzione MOV AL, OP1, gli 8 bit inferiori del registro AX conterranno il valore immagazzinato nella locazione di memoria OP1.

```
                      1 ;
                      2 ; PROGRAMMA PER ADDIZIONARE 2 NUMERI DI 8 BIT
                      3 ;
                      4  ORG 0200H
                      5 ;
                      6 ADR1          EQU 0300H
                      7 ADR2          EQU 0320H
                      8 ADR3          EQU 0345H
                      9 ;
0200 A00003          10  MOV AL,ADR1     ;Carica AL col dato ad ADR1
0203 02062003        11  ADD AL,ADR2     ;Somma AL col dato ad ADR2
0207 A24503          12  MOV ADR3,AL     ;Memorizza il risultato all'indirizzo ADR3
                     13 ;
```

Figura 5.3 — Questo è un programma assemblato dell'8086/8088 per addizionare due numeri di 8 bit.

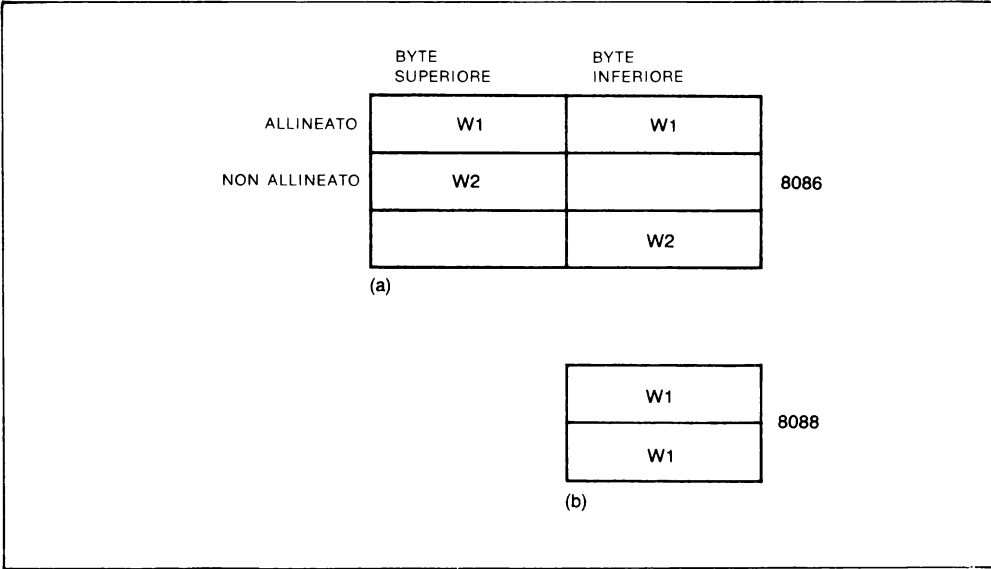



Figura 5.4a — I dati di 16 bit in un sistema 8086 possono essere allineati o non allineati.

Figura 5.4b — I dati di 16 bit in un sistema 8088 richiedono due byte contigui di memoria.

Perciò una parola (16 bit) richiederà 2 locazioni di memoria. La Figura 5.4 mostra come i dati sono oranizzati per le operazioni a 16 bit nella memoria del sistema 8086/8088. Un programma per sommare 2 numeri di 16 bit è mostrato nella Figura 5.5.

```
                1 ;
                2 ; PROGRAMMA PER ADDIZIONARE 2 NUMERI DI 16 BIT
                3 ;
                4  ORG 0250H
                5 ;
                6 ADR1          EQU 0300H
                7 ADR2          EQU 0320H
                8 ADR3          EQU 0345H
                9 ;
0250 A10003    10  MOV AX,ADR1    ;Carica AX col dato ad ADR1
0253 03062003  11  ADD AX,ADR2    ;Somma AX col dato ad ADR2
0257 A34503    12  MOV ADR3,AX    ;Memorizza il risultato all'indirizzo ADR3
               13 ;
```

Figura 5.5 — Questo programma assemblato dell'8086/8088 per addizionare 2 numeri di 16 bit è simile al programma della Figura 5.3, eccetto che il registro AX viene usato al posto del registro AL.

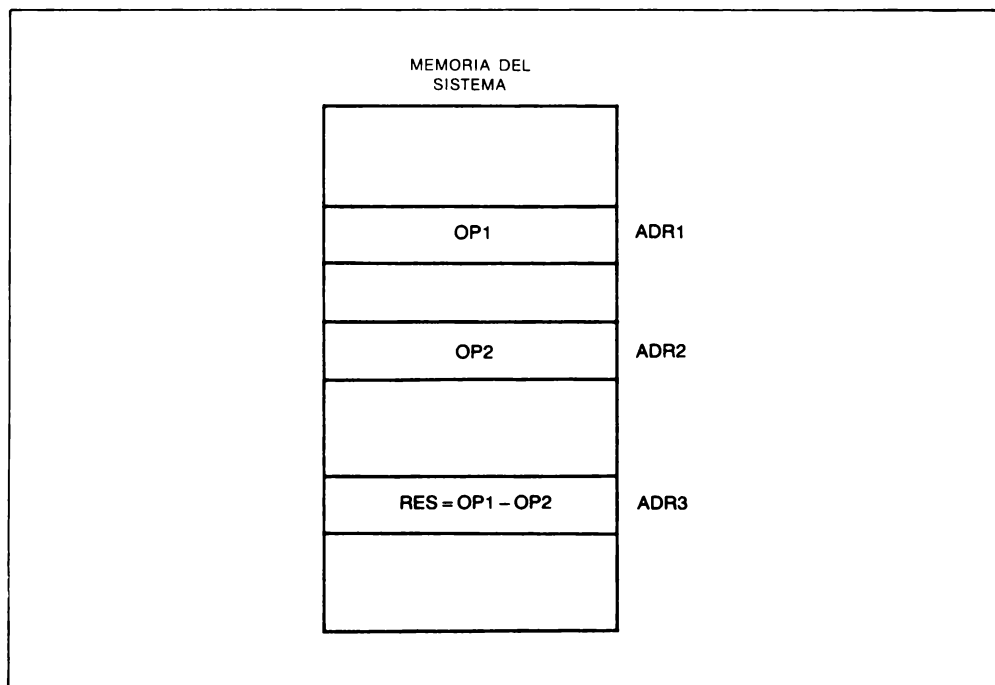



Figura 5.6 — Questo diagramma della memoria mostra come appariranno gli operandi (OP1 e OP2) in un esempio di sottrazione.

Sottrazione a 16 bit

La sottrazione di 2 numeri a 16 bit con l'8086/8088 è simile all'addizione a 16 bit risolta precedentemente.

Usando lo stesso esempio noi ora sottrarremo OP2 da OP1 e metteremo il risultato in ADR3. I dati risiederanno nella memoria nel modo illustrato nella Figura 5.6. Un programma per fare una sottrazione a 16 bit è mostrato in Figura 5.7.

ARITMETICA BCD

Nel Capitolo 1 abbiamo discusso il concetto di aritmetica BCD. Si ricordi che la rappresentazione BCD viene usata nelle applicazioni in cui è obbligatorio che si trattenga ogni cifra significativa del risultato.

Addizione BCD ad 8 bit

Nella notazione BCD per memorizzare una cifra decimale (0-9) viene usato un nibble (4 bit). Come risultato ogni byte (8 bit) può contenere 2 cifre BCD (questo è chiamato BCD impaccato). Vediamo ora come si opera in BCD. Sommiamo 2 byte ciascuno contenente 2 cifre BCD. Uno schema della configurazione della memoria in questo esempio è dato nella Figura 5.8.

```
                  1 ;
                  2 ; PROGRAMMA PER SOTTRARRE 2 NUMERI DI 16 BIT
                  3 ;
                  4 ORG 0250H
                  5 ;
                  6 ADR1        EQU 0300H
                  7 ADR2        EQU 0320H
                  8 ADR3        EQU 0345H
                  9 ;
0250 A10003      10 MOV AX,ADR1   ;Carica AX col dato ad ADR1
0253 2B062003    11 SUB AX,ADR2   ;Sottrae da AX il dato in ADR2
0257 A34503      12 MOV ADR3,AX   ;Memorizza il risultato all'indirizzo ADR3
                 13 ;
```

Figura 5.7 — Questo è un programma assemblato dell'8086/8088 per eseguire una sottrazione a 16 bit.

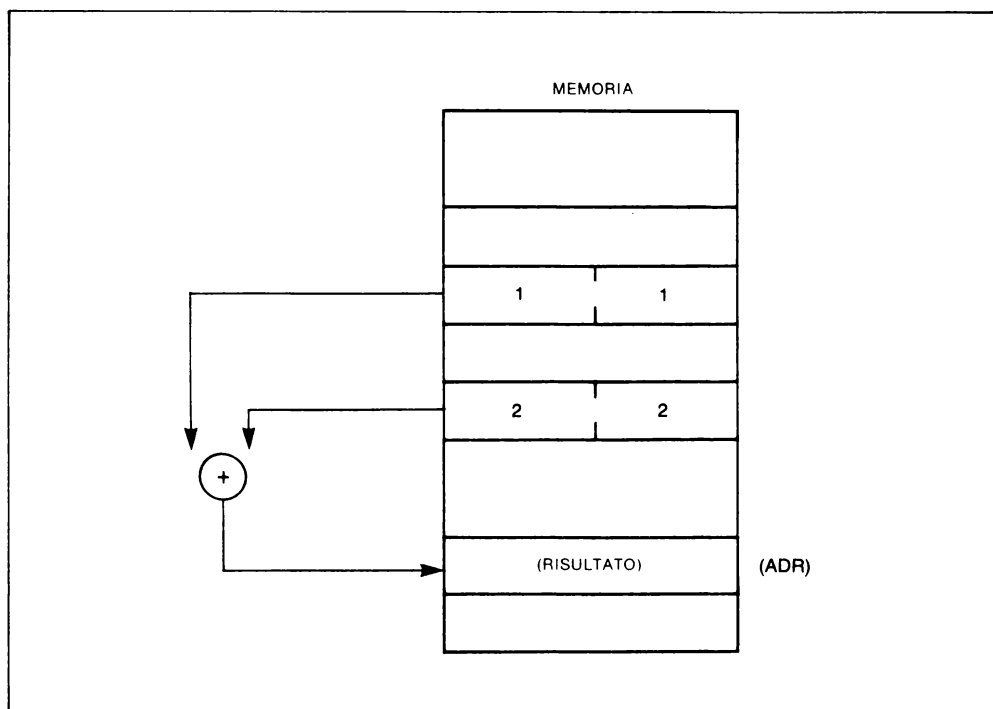



Figura 5.8 — Questo diagramma della memoria mostra la memorizzazione di due byte BCD impaccati prima e dopo l'addizione.

Prima di scrivere qualsiasi programma per eseguire l'addizione BCD, lavoriamo su alcuni esempi numerici. Questo ci aiuterà a definire qualsiasi problema che possa sorgere da questo tipo di addizione. Sommiamo dapprima 01 e 02

01 è rappresentato da	00000001
02 è rappresentato da	00000010
Il risultato è	00000011

Questo risultato è la rappresentazione BCD per 03 (se non si è sicuri dell'equivalente BCD si consulti la tabella di conversione data nell'appendice C. Tutto funziona abbastanza semplicemente in questo caso; proviamone allora un altro. Sommiamo 8 e 3:

08 è rappresentato da	00001000
03 è rappresentato da	00000011

Se si è ottenuto 00001011 come risultato significa che è stata eseguita la somma binaria di 8 e 3. Questo numero è effettiva-

mente 11 in binario. Sfortunatamente, 1011 è un codice scorretto in BCD. La rappresentazione BCD di 11 è 00010001. Questa differenza deriva dal fatto che la rappresentazione BCD usa solo le prime 10 combinazioni di 4 cifre per codificare i simboli decimali 0-9. Perciò, le rimanenti 6 possibili combinazioni di 4 cifre rimangono inutilizzate. In altre parole, ogni qualvolta la somma di due numeri BCD è maggiore di 9 si deve aggiungere 6 al risultato per saltare i sei codici inutilizzati.

Correzione del risultato di una somma BCD

Il seguente esempio mostra come correggere l'inconveniente appena discusso. Con la semplice somma di 6 all'equivalente binario di 11 (la somma che otteniamo addizionando 8 e 3) potremo ottenere il risultato corretto:

```
1011        risultato illegale in BCD
0110        + 6
--------
00010001
```

Questo risultato è uguale a 11 in BCD. Ora bbiamo la risposta corretta al problema dell'addizione BCD di 8 e 3.

Questo esempio illustra una delle difficoltà di base del metodo BCD: si devono compensare i 6 codici mancanti. C'è, tuttavia, una speciale istruzione di adattamento dell'addizione decimale (DAA) che aggiusterà automaticamente il risultato dell'addizione binaria (questa istruzione aggiungerà 6 se il risultato è maggiore di 9).

Useremo ora questo esempio per illustrare un altro punto riguardante l'aritmetica BCD. Quando aggiungiamo 6 al risultato di 11 viene generato un riporto dal nibble (semi byte) inferiore al nibble superiore. Questo riporto interno deve essere preso in considerazione e sommato alla seconda cifra BCD. Sebbene l'istruzione di addizione si occupi automaticamente di questo problema, conviene essere in grado di controllare realmente questo riporto interno da bit3 a bit4. Questo può essere fatto testando il flag AF (Nota: questo riporto è talvolta chiamato semi-riporto o riporto ausiliario).

Come esempio ecco le istruzioni del programma per addizionare i 2 numeri BCD 11 e 22.

```
MOV  AL,11H      11 nel registro AL
ADD  AL,22H      Somma 11 a 22 e salva il risultato in AL
DAA              Adattamento decimale del risultato
MOV  ADR3,AL     Immagazzina il risultato in memoria
```

Nota: l'H che segue i numeri 11 e 22 indica che questi numeri sono espressi in notazione esadecimale, non in decimale). Il registro AL è usato poichè l'istruzione DAA richiede che l'operando sia nel registro AL.

Quando e come usare l'istruzione di adattamento DAA

Questo programma è simile a quello dato per l'addizione binaria ad 8 bit. Tuttavia usa una nuova istruzione: DAA (adattamento decimale per l'addizione). (Si consulti il Capitolo 4 per ulteriori informazioni sull'istruzione DAA).

Vediamo ora esattamente cosa fa in questo programma l'istruzione DAA. La prima cosa che accade è che 11 viene sommato a 22. Questo apparirà nel registro AL come segue:

```
  00010001 (11)
+ 00100010 (22)
---------------
= 00110011 (33)
```

Poichè il numero nei 4 bit inferiori del risultato non è maggiore di 9 e poichè non c'era nessun riporto dal bit3 al bit4, l'istruzione DAA non ha fatto niente nei quattro bit inferiori del risultato. (AF = 0). Inoltre, non c'è bisogno di operare sui 4 bit superiori del risultato. Perciò in questo caso l'istruzione DAA non ha cambiato in nessun modo il risultato.

Prendiamo ora un altro esempio. Addizioneremo i numeri 22 e 39:

```
  00100010 (22)
+ 00111001 (39)
---------------
  01011011 (5?)
```

100 è un codice BCD illegale, poichè è maggiore di 9.

Il risultato deve essere corretto. Possiamo usare l'istruzione DAA che aggiungerà 6 ai 4 bit inferiori e genererà un riporto nei successivi 4 bit. Questo darà il seguente risultato:

```
  01011011 (5?)
+ 00000110 (06)
---------------
  01100001 (61)
```

Questo è il risultato corretto per l'addizione di 22 e 39.

In questo caso è stato necessario usare l'istruzione DAA per generare la soluzione corretta.

Sottrazione BCD

Effettuare la sottrazione BCD con l'8086/8088 è altrettanto semplice quanto effettuare l'addizione BCD.

Questo è dovuto al fatto che la CPU ha un'istruzione speciale DAS (adattamento decimale per la sottrazione). (Consultare il Capitolo 4 per questa istruzione). Ecco un programma per sottrarre 2 numeri validi BCD impaccati.

```
MOV  AL,42H      carica AL con 42 BCD
SUB  AL,23H      sottrae 23 da 42
DAS              adattamento decimale per la sottrazione
MOV  ADDR3,AL    immagazzina il risultato in memoria
```

Finora abbiamo mostrato esempi di addizione e sottrazione usando differenti istruzioni per l'8086/8088. Passiamo ora ad esaminare le operazioni aritmetiche di moltiplicazione e divisione.

Moltiplicazione

Cominciamo il nostro studio sulla moltiplicazione esaminando un problema semplice di moltiplicazione decimale. Moltiplichiamo 12 per 23:

```
    12   moltiplicando
  × 23   moltiplicatore
  ----
    36   prodotto parziale
  + 24
  ----
   276   risultato finale
```

La moltiplicazione è effettuata dapprima con la moltiplicazione della cifra più a destra del moltiplicatore per il moltiplicando, cioè 3×12 (il prodotto parziale è 36), e poi moltiplicando la cifra seguente del moltiplicatore (cioè 2) per 12; sommando poi questi due risultati si ottiene il risultato finale.

C'è tuttavia ancora un'operazione: o 24 deve essere spostato a sinistra di una posizione (di una cifra) o il prodotto parziale (36) deve essere spostato a destra di una posizione. I due numeri sono poi sommati. La somma è 276. Esaminiamo ora una moltiplicazione binaria. Questa viene effettuata come una moltiplicazione decimale. Moltiplichiamo 5 per 3:

```
 (5)     101   moltiplicando
 (3)   × 011   moltiplicatore
       -----
         101   prodotto parziale
        101
      + 000
       -----
(15)   01111   risultato finale
```

Da questo esempio possiamo vedere che la moltiplicazione binaria è molto simile alla moltiplicazione decimale.

Siamo fortunati che l'8086/8088 sia fornito di un'istruzione

che effettuerà per noi questa moltiplicazione. I microprocessori meno recenti, come pure i microprocessori a 8 bit, non hanno l'istruzione di moltiplicazione. Perciò la moltiplicazione è effettuata con un breve programma. Infatti, la sola aritmetica che questi microprocessori possono compiere è l'addizione e lo spostamento (shift).

Esaminiamo ora come la moltiplicazione viene effettuata sull'8086/8088.

Moltiplicazione di numeri di 16 bit

Con l'8086/8088 è possibile effettuare la moltiplicazione direttamente su due numeri di 16 bit. Quando l'operazione è completa, il risultato è messo in 2 registri a 16 bit. Questo è necessario perchè $2^{16} * 2^{16}$ è uguale a 2^{32}. Perciò, è necessario un registro a 32 bit per trattenere la massima risposta possibile ottenuta con la moltiplicazione di due numeri di 16 bit.

Registri da utilizzare per effettuare una moltiplicazione

Con l'8086/8088, uno dei numeri da moltiplicare è immagazzinato nel registro AX.

Il risultato a doppia lunghezza (32 bit) è immagazzinato nei registri AX e DX con i 16 bit più significativi nel registro DX.

Se la moltiplicazione deve essere un'operazione ad 8 bit, uno dei due numeri deve stare nel registro AL. Il risultato a 16

```
                 1 ;
                 2 ; PROGRAMMA 8086/8080 PER MOLTIPLICARE
                 3 ; DUE NUMERI A 16 BIT. I NUMERI SONO
                 4 ; 2345x5378=12611410=00C06F52
                 5 ;
                 6  ORG 250H
                 7 ;
                 8 ADR3        EQU 300H
                 9 ADR4        EQU ADR3+2
                10 ;
0250 B82909     11  MOV AX,#2345   ;2345 NEL REGISTRO AX
0253 B90215     12  MOV CX,#5378
0256 F7E1       13  MUL CX         ;MUL AX x (CX)5378
0258 A30003     14  MOV ADR3,AX    ;MEMORIZZA AX IN ADR3 (16 BIT INFERIORI)
025B 89160203   15  MOV ADR4,DX    ;MEMORIZZA DX IN ADR4 (16 BIT SUPERIORI)
                16 ;
```

Figura 5.9 — Questo è un programma assemblato dell'8086/8088 per moltiplicare 2 numeri di 16 bit.

bit è riportato nel registro AX, con il byte più significativo in AH. La Figura 5.9 mostra un programma per moltiplicare due numeri di 16 bit e memorizzare il risultato nelle locazioni di memoria ADR3 e ADR4.

Divisione binaria

Cominciamo la nostra discussione sulla divisione binaria esaminando la divisione decimale. Dividiamo 254 per 12:

```
                21  (quoziente)
(divisore) 12 ⌉254  (dividendo)
              -24
              ----
               14
              -12
              ----
                2     (resto)
```

Esaminando questo problema, possiamo vedere che la divisione è eseguita sottraendo il multiplo più grande possibile del divisore dalla cifra più a sinistra del dividendo (questo genera un nuovo dividendo di 14). Il moltiplicatore del divisore ora diventa la seconda cifra del quoziente. Infine il resto e' il risultato dell'ultima sottrazione possibile.

Per trovare il più grosso multiplo del divisore che può essere sottratto dal dividendo, debbiamo fare prove di confronti e sottrazioni. Si deve notare che nel determinare la prima cifra del quoziente il numero effettivo è 20 e non 2; e il numero sottratto dal dividendo è 240 e non 24.

Lasciando da parte gli zeri siamo in grado di fare una notazione conveniente ma non dobbiamo perdere di vista ciò che sta realmente accadendo nel procedimento.

La divisione binaria è eseguita nello stesso modo, come dimostra il seguente esempio:

```
               0011 (quoziente)
(divisore) 11 ⌉1010 (dividendo)
             - 11
             -----
               100
             - 11
             -----
                 1 (resto)
```

Questo esempio ha seguito la stessa procedura dell'esempio decimale presentato prima. Come possiamo vedere la divisione ha come risultato un quoziente ed un resto!

Divisione con l'8086/8088

Registri da utilizzare per eseguire una divisione

L'8086/8088 fornisce un'istruzione di divisione che effettuerà per noi una completa operazione di divisione. Il dividendo per la CPU è immagazzinato nei registri AX e DX questo significa che il dividendo è una quantità di 32 bit con i 16 bit più significativi situati nel registro DX.

Per questa divisione il divisore è una quantità a 16 bit.

Quando l'operazione è terminata il quoziente a 16 bit è contenuto nel registro AX. Il resto a 16 bit è contenuto nel registro DX. Il resto a 16 bit viene ritornato nel registro DX.

Quando la divisione viene eseguita su quantità che occupano un byte, il dividendo a 16 bit è posto nel registro AX. A conclusione dell'operazione il quoziente a 8 bit è memorizzato nel registro AL, mentre il resto viene posto nel registro AH.

La Figura 5.10 mostra un programma per l'8086/8088 che dividerà due numeri e salverà in memoria il quoziente ed il resto.

SUBROUTINES

In questo paragrafo esamineremo come opera l'8086/8088 con le subroutines. Cominceremo col dare una definizione ge-

```
                  1 ;
                  2 ; PROGRAMMA PER DIVIDERE 2 NUMERI
                  3 ; IL PROBLEMA SARÀ DIVIDERE 450 PER 20
                  4 ; IL RISULTATO SARÀ 22 CON RESTO 10
                  5 ;
                  6  ORG 250H
                  7 ;
                  8 ADR3    EQU 300H        ;2 BYTE PER QUOZIENTE
                  9 ADR4    EQU ADR3+2      ;2 BYTE PER RESTO
                 10 ;
0250 31D2        11  XOR DX,DX        ;AZZERA IL REGISTRO DX
0252 B8C201      12  MOV AX,#450      ;450 È CARICATO IN AX
0255 B91600      13  MOV CX,#22       ;22 È CARICATO IN CX
0258 F7F1        14  DIV CX           ;DIV DXAX BY CX(22)
025A A30003      15  MOV ADR3,AX      ;IMMAGAZZINARE QUOZIENTE IN MEMORIA
025D 89160203    16  MOV ADR4,DX      ;IMMAGAZZINARE RESTO IN MEMORIA
                 17 ;
```

Figura 5.10 — Questo è un programma assemblato dell'8086/8088 per dividere due numeri di 16 bit.

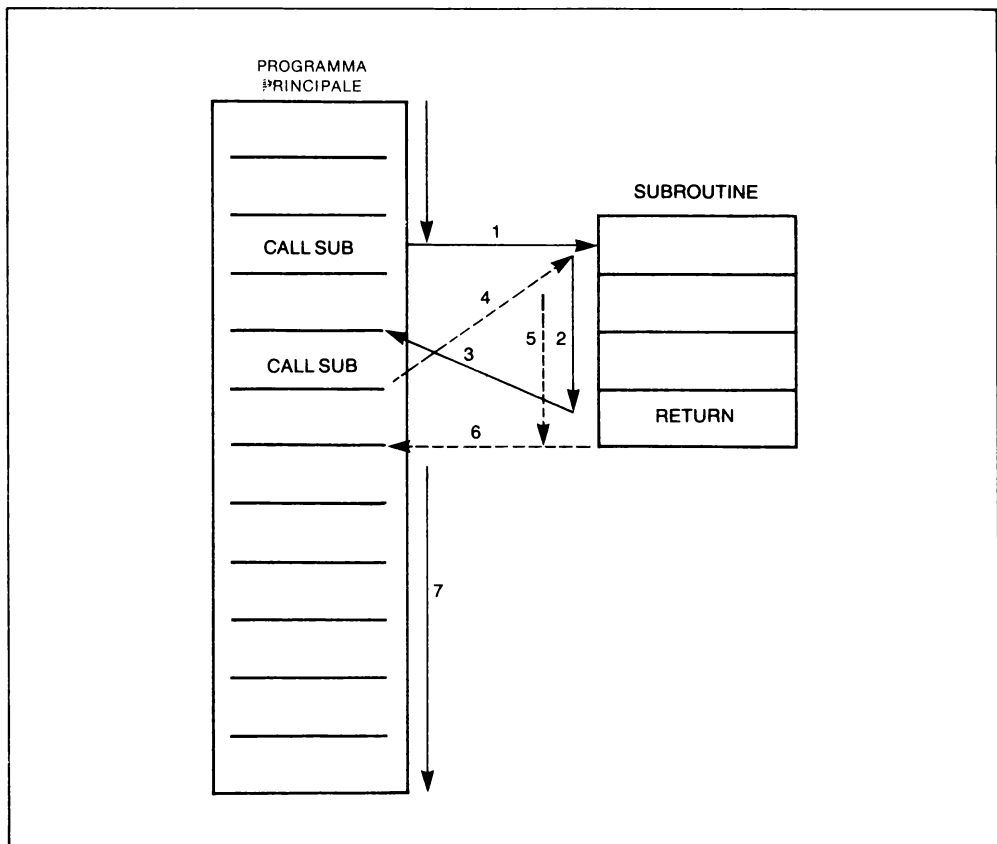



Figura 5.11 — Il diagramma mostra come una subroutine possa essere chiamata da diverse locazioni nel programma principale. Le frecce 1, 2 e 3 mostrano il cammino dell'esecuzione per la prima CALL SUB; le frecce tratteggiate 4, 5 e 6 mostrano il cammino per la seconda CALL SUB.

nerale di subroutine; procederemo poi nel dettaglio dell'uso delle subroutines con la CPU.

Cos'è una subroutine

Concettualmente una subroutine è semplicemente un blocco o una parte di istruzioni chiamate dal programmatore. Una subroutine è eseguita quando il programma principale esegue l'istruzione speciale CALL; la subroutine è poi terminata con una speciale istruzione chiamata RETURN. Illustriamo ora l'uso di una subroutine per dimostrarne il valore. La Figura 5.11 illustra come viene usata una subroutine. Il programma principale appare sulla sinistra della figura e la subroutine appare, simbolicamente, sulla destra. Esaminiamo ora in che modo funziona una subroutine. In questo programma, le righe del programma principale sono eseguite in sequenza fino a quando viene incontrata una istruzione CALL SUBROUTINE. L'e-

Uso di una subroutine

secuzione di questa istruzione ha come risultato un trasferimento alla sezione subroutine del programma. (Così la prossima istruzione che deve essere eseguita dopo la CALL SUBROUTINE è la prima istruzione della subroutine). Questo è illustrato dalla freccia 1 in Figura 5.11 (In questo esempio, la sezione del programma che stiamo usando come subroutine viene eseguita come qualsiasi altro programma, come indicato dalla freccia numero 2. In altre parole la subroutine non contiene nessuna istruzione CALL SUBROUTINE [descritta più avanti]).

L'ultima istruzione della subroutine è un RETURN. Questa speciale istruzione fa si che la CPU ritorni al programma principale. Quando la CPU ritorna al programma principale eseguirà l'istruzione che viene immediatamente dopo l'istruzione CALL SUBROUTINE che aveva inizialmente obbligato la CPU ad andare alla sezione subroutine del programma. Questo è mostrato dalla freccia 3 in Figura 5.11.

Successivamente, una seconda istruzione CALL SUBROUTINE appare nel corpo del programma principale. Avverrà quindi un nuovo trasferimento come mostra la freccia 4. Questo significa che il corpo della subroutine viene eseguito di nuovo, dopo l'istruzione di CALL SUBROUTINE.

Tutte le volte che si incontra un RETURN dentro una subroutine, questo riporta alla istruzione che segue l'ultima CALL SUBROUTINE che è stata eseguita. Questo è illustrato dalla freccia 6.

Dopo il ritorno al programma principale, l'esecuzione procede normalmente, come mostrato dalla freccia 7.

Vantaggi derivanti dall'uso di una subroutine

L'effetto delle due istruzioni CALL SUBROUTINE e RETURN dovrebbe ora essere chiaro. Qual'è l'uso di una subroutine? Il pregio maggiore di una subroutine è che può essere chiamata in ogni punto del programma principale e le istruzioni interne alla subroutine possono essere usate ripetutamente senza dover essere riscritte. Un vantaggio di questo tipo di approccio è che si risparmia spazio in memoria perchè la subroutine non deve essere riscritta ogni volta. Un altro vantaggio è che il programmatore ha bisogno di progettare una specifica subroutine una sola volta, dato che può essere riutilizzata più volte. Questa è una importante semplificazione del processo di progettazione del programma.

Implementazione del meccanismo di subroutine

Discuteremo ora come le due istruzioni CALL SUBROUTINE e RETURN sono implementate dentro un microprocesso-

re. Dapprima discuteremo la CALL SUBROUTINE. L'istruzione CALL SUBROUTINE fa si che un nuovo indirizzo venga messo nel registro IP della CPU. Ricordiamo che il registro IP determina l'indirizzo di memoria da cui deve essere prelevata la successiva istruzione da eseguire. In altre parole, viene caricato nel registro IP l'indirizzo di partenza della subroutine. Ma ciò è sufficiente?

Per rispondere a questa domanda consideriamo l'altra istruzione: RETURN. Questa istruzione provoca un ritorno all'istruzione del programma principale che segue la CALL SUBROUTINE. Una azione come questa è possibile solo se l'indirizzo di memoria dell'istruzione che segue la CALL SUBROUTINE è stato conservato da qualche parte.

Il problema seguente è conservare da qualche parte questo indirizzo di ritorno: questo indirizzo deve essere sempre conservato in una locazione in cui non possa essere cancellato.

Operazioni eseguite dalle istruzioni CALL e RETURN

Quando la CPU incontra una istruzione di CALL SUBROUTINE, il valore del registro IP viene automaticamente salvato. A questo punto il registro IP è uguale all'indirizzo offset della successiva istruzione che dovrebbe essere eseguita. In questo caso, è l'indirizzo di memoria che segue direttamente l'istruzione di CALL SUBROUTINE.

Successivamente la CPU comincia l'esecuzione all'indirizzo della subroutine. Quando l'istruzione RETURN è eseguita nella subroutine, il valore salvato dell'indirizzo viene ripreso dalla locazione di memoria dove era conservato. Il conservare e riprendere l'indirizzo dell'istruzione che segue l'istruzione CALL SUBROUTINE è fatto automaticamente dalla CPU. L'istruzione CALL SUBROUTINE conserva l'indirizzo e l'istruzione RETURN rimette l'indirizzo in IP.

Abbiamo precedentemente affermato che l'indirizzo è conservato in una specifica locazione di memoria. La domanda è: "dove è conservato l'indirizzo nella memoria?" È conservato nella zona di memoria definita come stack. Lo stack è una speciale parte della memoria riservata per certe operazioni della CPU.

Mostreremo queste operazioni nei prossimi capitoli di questo testo. Nel Capitolo 4 abbiamo presentato le istruzioni PUSH e POP che usano lo stack del sistema. Quando si incontra una istruzione di CALL SUBROUTINE, l'indirizzo della successiva istruzione che deve essere eseguita viene memorizzato nello stack. Quello è il posto dove è conservato. Quando la CPU incontra una istruzione RETURN, l'indirizzo conservato viene automaticamente fatto uscire dallo stack. Questo indirizzo prelevato è poi usato come indirizzo per l'istruzione seguente.

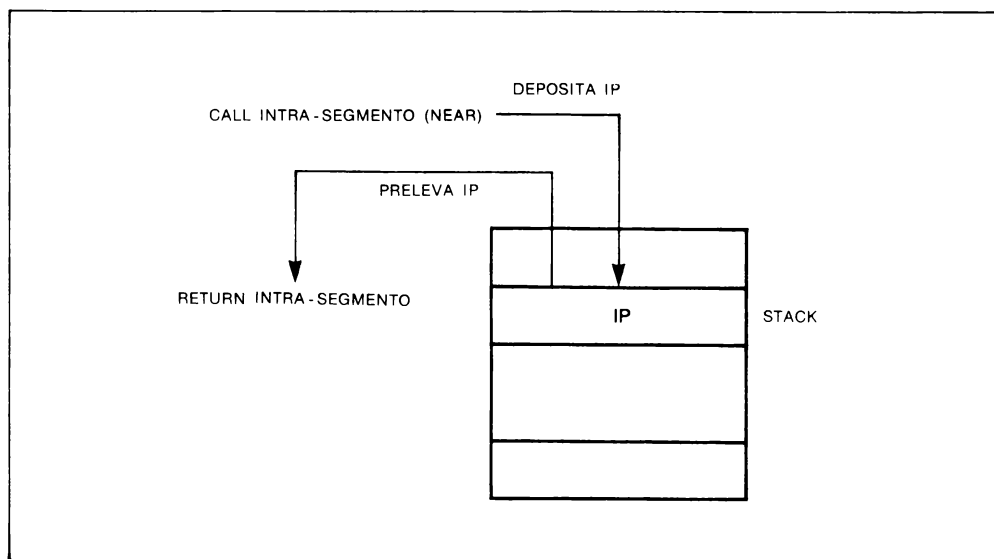



Figura 5.12 — Una CALL intra-segmento (all'interno di un blocco di dati di 64K) salva il registro IP sullo stack del sistema. Il RETURN intra-segmento corrispondente preleva il registro IP dallo stack.

CALL intrasegmento

Ci sono due tipi principali di CALL SUBROUTINE: intrasegmento e intersegmento. Esaminiamole.

L'istruzione CALL intrasegmento fa si che la CPU esegua una subroutine che e' posta nello stesso blocco segmento di 64K di memoria.

Quando gli 8086/8088 incontrano questo tipo di CALL, il registro IP è posto nello stack. Quando si incontra un'istruzione RETURN il valore del registro IP è tolto dalla stack e messo di nuovo nel registro IP della CPU, come mostra la Figura 5.12

CALL intersegmento

Questa CALL fa si che l'8086/8088 esegua una subroutine che risiede fuori dal blocco segmento di 64K del codice. In questo caso sia il registro CS (segmento codice) che il registro IP sono salvati nello stack. Quando si incontra un'istruzione di ritorno, entrambi i valori di CS e IP conservati sono tolti dallo stack e inseriti nel registro interno della CPU. Questo è mostrato nel diagramma a blocchi della Figura 5.13

CALL-RETURN SBAGLIATO

Possiamo vedere dagli esempi precedenti di CALL intrasegmento e intersegmento che l'istruzione RETURN deve o prelevare i registri CS e IP o solo il registro IP. Per questo fatto ci deve essere più di un tipo di istruzione di RETURN. Esiste. Ricordate che nel Capitolo 4 c'erano RETURN intersegmento e intrasegmento. Questo significa che una subroutine deve essere sempre usata o come una routine intrasegmento o come una intersegmento. La Figura 5.14 mostra un abbinamento sbagliato CALL-RETURN.

SOMMARIO

In questo capitolo abbiamo presentato alcuni concetti basilari di programmazione. Questi comprendevano l'addizione, la sottrazione, la moltiplicazione e la divisione binarie, ed anche l'aritmetica BCD.

Abbiamo scritto diversi programmi e spiegato come l'8086/8088 realizza questi concetti.

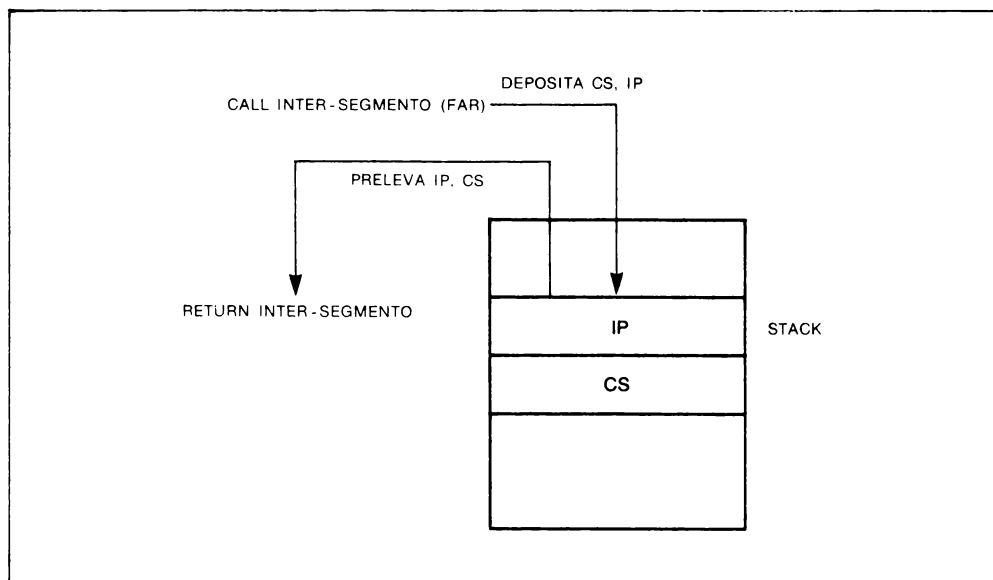



Figura 5.13 — Una CALL inter-segmento (al di fuori di un blocco di dati di 64K) salva entrambi i registri CS e IP sullo stack del sistema. Un RETURN inter-segmento preleva entrambi questi registri dallo stack.

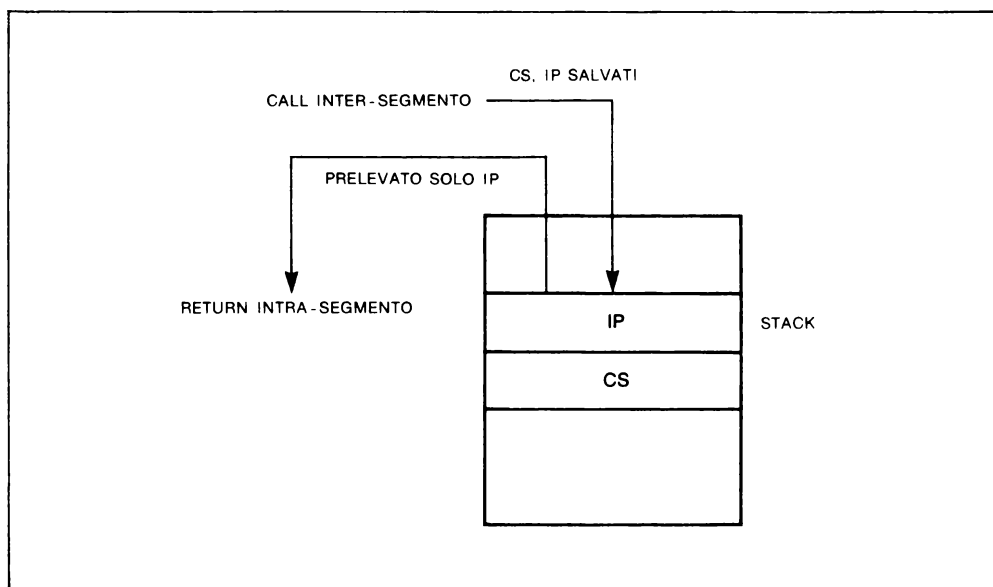



Figura 5.14 — Questa figura mostra un abbinamento CALL-RETURN sbagliato. Nel diagramma si vede che la CALL inter-segmento deposita sullo stack sia IP che CS. Un RETURN intra-segmento, invece, preleva dallo stack solo il registro IP. Questo provoca un errore nell'abbinamento CALL-RETURN in conseguenza del quale il programma fallisce, cioè non prosegue correttamente la sua esecuzione.

Verso la fine di questo capitolo abbiamo discusso l'uso delle subroutine nella programmazione. Abbiamo esaminato come l'8086 e l'8088 implementano le CALL SUBROUTINE e i RETURN e abbiamo dato un'occhiata ai CALL e RETURN intrasegmento e intersegmento. Troveremo utile quest'ultima informazione quando discuteremo le diverse applicazioni di programmazione che riguardano l'8086/8088 nei capitoli successivi.

CAPITOLO 6

INTERRUZIONI PER L'8086/8088

INTRODUZIONE

In questo capitolo discuteremo l'argomento generale delle interruzioni dei microprocessori 8086/8088. Dapprima introdurremo il concetto di interruzione, mostreremo poi come ogni diverso tipo di interruzione venga gestita elettricamente dall'8086/8088. Inoltre esamineremo utili esempi di software e discuteremo ciò che realmente avviene nella CPU durante un'operazione di interruzione.

CHE COS'È UN'INTERRUZIONE?

Il modo migliore per dare un esempio di interruzione è il seguente: immaginatevi di star chiaccherando con una persona. Una seconda persona si avvicina e vi chiama – richiamando la vostra attenzione –. Ci possono essere 3 possibilità di risposta a questa richiesta esterna:

1) Si può ignorare completamente la seconda persona e continuare la conversazione con la prima.
2) Si può arrivare ad un punto conveniente per interrompere la conversazione e poi rivolgere l'attenzione alla seconda persona.
3) Si può smettere immediatamente la conversazione con la prima persona e conversare con la seconda Ad ogni modo, è probabile che quando avrete finito di parlare con la seconda persona vorrete continuare a parlare con la prima.

Benchè questo scenario possa sembrare semplicistico, presenta accuratamente il concetto di interruzione in un sistema di microprocessori.

Immaginate di essere la CPU 8086/8088 e pensate che la prima persona sia il programma principale che deve essere eseguito, e la seconda persona sia una richiesta esterna di interruzione. (cioè, che sia qualcosa di hardware nel sistema che ri-

chiede l'attenzione della CPU). La CPU 8086/8088 deve trattare in qualche modo questa richiesta.

Questo può essere fatto in diversi modi. Le possibilità sopra elencate sono le più comuni.

Dopo questa introduzione generica sulle interruzioni concentriamoci ora sui dettagli delle interruzioni per l'8086/8088.

Da dove proviene la richiesta di un'interruzione esterna?

Un sistema basato su microprocessore può essere composto da diverse componenti hardware: la stampante, il drive dell'hard disk o del floppy disk, un CRT, un timer, o un convertitore da digitale ad analogico (DAC) solo per nominarne alcuni. La maggior parte di questi componenti hardware esterni hanno bisogno dell'attenzione della CPU solo in certi momenti. In altri momenti possono funzionare da soli.

Un esempio pratico di interruzione

Per esempio supponiamo che il vostro sistema abbia un orologio come strumento esterno hardware che deve mostrare l'ora del giorno sul video CRT.
L'orologio hardware richiede che la CPU legga l'ora una sola volta ogni secondo e la stampi sul video CRT. Tutte le altre volte la CPU è disponibile per eseguire altri compiti richiesti.

Il punto principale qui è che l'hardware esterno dell'orologio non richiede l'attenzione costante della CPU.

L'orologio hardware richiede elettricamente alla CPU di essere letto solo quando è necessario. Un modo per realizzare questo consiste nell'utilizzare il sistema di interruzione della CPU, per mezzo del quale la CPU riceve una volta al secondo una richiesta elettrica esterna di interruzione da parte dell'orologio hardware. A questo punto la CPU blocca tutto quello che sta facendo e legge l'ora dell'orologio. Dopo aver letto l'orologio e aver mostrato sul video l'ora, la CPU riprenderà l'esecuzione del programma che stava girando prima dell'interruzione.

Questo semplice esempio di interruzione vi aiuterà a rispondere alla domanda iniziale, cioè da dove arrivano le richieste di interruzione. La risposta è che queste arrivano dall'hardware del sistema microprocessore.

Interruzioni non mascherabili (NMI)

Le linee di ingresso delle interruzioni

Come mostrato in Figura 6.1, ci sono 3 linee principali di ingresso delle interruzioni nei microprocesori 8086/8088: INTR,NMI e RESET. Ciascuno di questi input può causare una

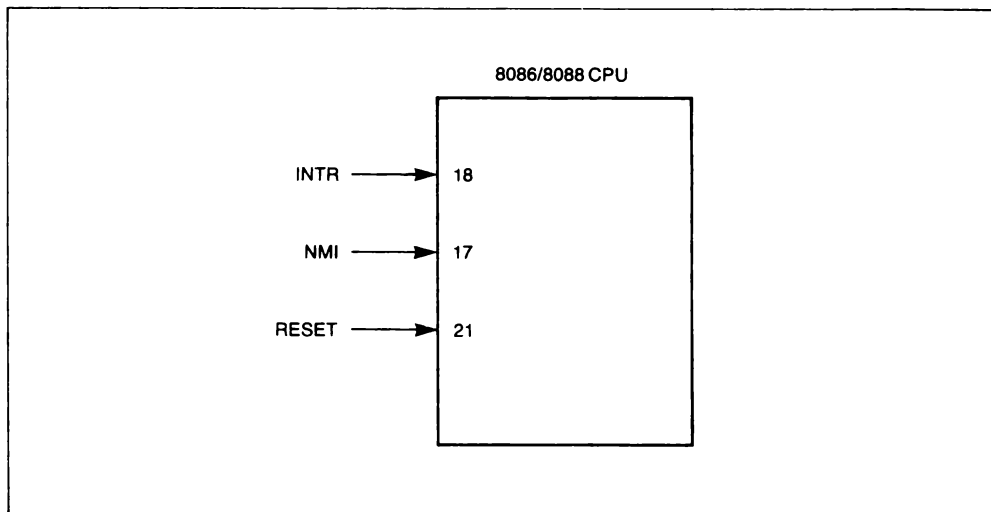



Figura 6.1 — Questo diagramma mostra gli ingressi delle interruzioni INTR, NMI e RESET per l'8086/8088.

richiesta esterna di interruzione. Inizieremo la nostra discussione dall'NMI che è l'ingresso delle richieste di interruzioni non mascherabili. Tratteremo i diversi punti importanti che riguardano le interruzioni nell'8086/8088. Molti di questi punti sono comuni agli altri tipi ingressi delle interruzioni.

Cominceremo la nostra discussione con l'interruzione non mascherabile perchè opera in un solo modo.

Il termine "interruzione non mascherabile" significa che questa interruzione deve essere sempre riconosciuta dall'8086/8088 appena è elettricamente possibile. (Questa era la terza possibilità nella nostra lista precedente). Dato che l'8086/8088 ha un'interruzione non mascherabile, ne segue che ci deve essere anche un ingresso per le interruzioni mascherabili. Questo è l'ingresso INTR. Discuteremo di questo più avanti nel capitolo.

Riconoscimento di una richiesta NMI

Supponiamo che esternamente sia stata effettuata una richiesta di interruzione NMI. La successiva domanda è "quando la richiesta NMI verrà riconosciuta elettricamente dalla CPU?". Generalmente viene riconosciuta alla fine dell'attuale ciclo di istruzione; cioè quando l'istruzione attualmente in esecuzione è completata. Nota che ci sono alcune istruzioni sull'8086/8088 che possono richiedere parecchio tempo per l'esecuzione(es. le istruzioni di divisione e di moltiplicazione).

Tempo di latenza di un'interruzione

Il tempo richiesto perchè la CPU risponda elettricamente a una richiesta di interruzione è conosciuto come "latenza di interru-

zione''. L'effettiva quantità di tempo dipende da quanti cicli di clock mancano al termine dell'istruzione che è attualmente in esecuzione.

L'ATTIVITÀ DELLA CPU DURANTE LA RICHIESTA NMI

Durante un'operazione NMI la prima cosa che avviene è che i flag vengono memorizzati nell'area dello stack del sistema. Questa azione salva il valore attuale di tutti i flag del sistema. Poi la CPU azzera l'IF (flag di abilitazione delle interruzioni) e perciò disabilita qualsiasi interruzione dalla linea di ingresso INTR. Quando questo flag è azzerato, tutte le richieste di interruzione dall'ingresso INTR sono elettricamente ignorate. Viene poi azzerato un flag speciale dell'8086/8088 chiamato TF (flag di trap). Questo flag è usato quando l'8086/8088 è posta in modo ''single step'' (passo singolo). Perciò, azzerando questo flag, si ha come conseguenza che l'8086/8088 non viene più usato in modo single step.

La successiva azione intrapresa dalla CPU è quella di memorizzare i registri CS e IP nello stack del sistema. A questo punto nel corso del trattamento dell'NMI da parte della CPU, lo stack del sistema appare come mostra la Figura 6.2.

Infine, la CPU carica il registro IP con il valore a 16 bit che si trova all'indirizzo di memoria 00008H. Il registro CS è poi caricato con il valore a 16 bit collocato all'indirizzo 0000AH. Viene poi generato un nuovo indirizzo di memoria, usando i registri CS e IP appena caricati. Poi, la CPU fa partire l'esecuzione del codice individuato da questo nuovo indirizzo di memoria.

Gestione di una interruzione non mascherabile

Rivedendo quanto detto, i passi eseguiti quando la CPU gestisce una richiesta NMI sono:

1) Il registro FLAG è salvato nello stack del sistema

2) Viene disabilitato l'ingresso INTR

3) TF viene azzerato, non è possibile procedere in single step

4) Il registro CS viene salvato nello stack del sistema

5) Il registro IP viene salvato nello stack del sistema

6) Il registro IP viene caricato con i dati a 16 bit prelevati all'indirizzo di memoria 00008H

7) Il registro CS viene caricato con i dati a 16 bit prelevati all'indirizzo di memoria 0000AH

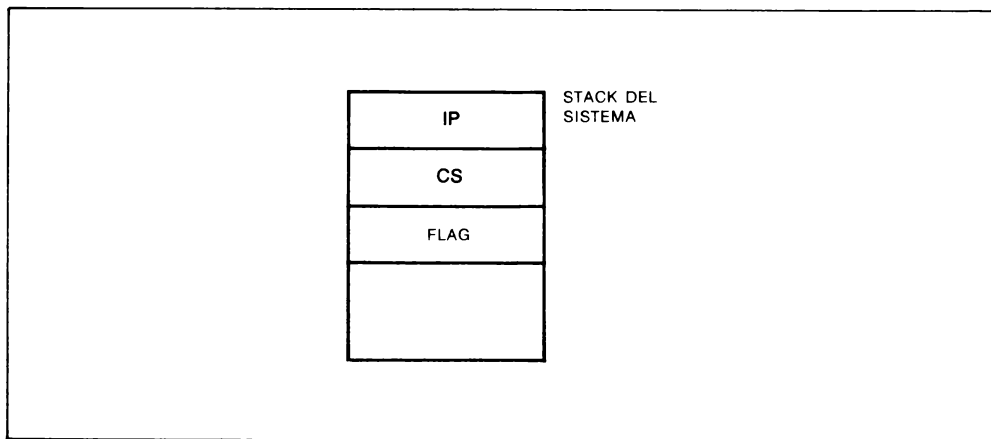



Figura 6.2 — Ecco come lo stack del sistema dopo che i registri FLAG, CS e IP sono stati salvati sullo stack durante la gestione di un'interruzione.

8) La CPU va a prendere la successiva istruzione dall'indirizzo a 20 bit generato con i nuovi valori dei registri CS e IP.

A questo punto la CPU sta eseguendo la routine di servizio di una interruzione per una richiesta di interruzione MNI.

LA TABELLA DELLE INTERRUZIONI

Prima di procedere con la nostra discussione sull'ingresso INTR, esaminiamo il concetto di tabella delle interruzioni. Nella nostra spiegazione dell'interruzione NMI abbiamo affermato che gli indirizzi 00008H e 0000AH sono usati per la memorizzazione dei registri CS e IP. I valori contenuti in questi registri sono usati per generare il nuovo indirizzo a 20 bit. Questi valori sono stati ottenuti da una sequenza di indirizzi chiamata tabella delle interruzioni. La Figura 6.3 mostra uno schema della tabella delle interruzioni per l'8086/8088.

Questa tabella occupa i primi 1024 byte della memoria del sistema; cioè gli indirizzi di memoria 00000-003FF. Ci sono 255 tipi differenti di interruzioni che possono essere trattati dall'8086/8088. Più avanti in questo capitolo discuteremo in dettaglio l'argomento dei tipi di interruzioni. Si imparerà che ad ogni tipo è data un'etichetta, dal tipo 0 al tipo 255.

Dimensioni e contenuti degli elementi della tabella delle interruzioni

Per ciascun tipo di interruzione (da 0 a 255) sono riservati nella tabella delle interruzioni 4 byte di memoria. Questi 4

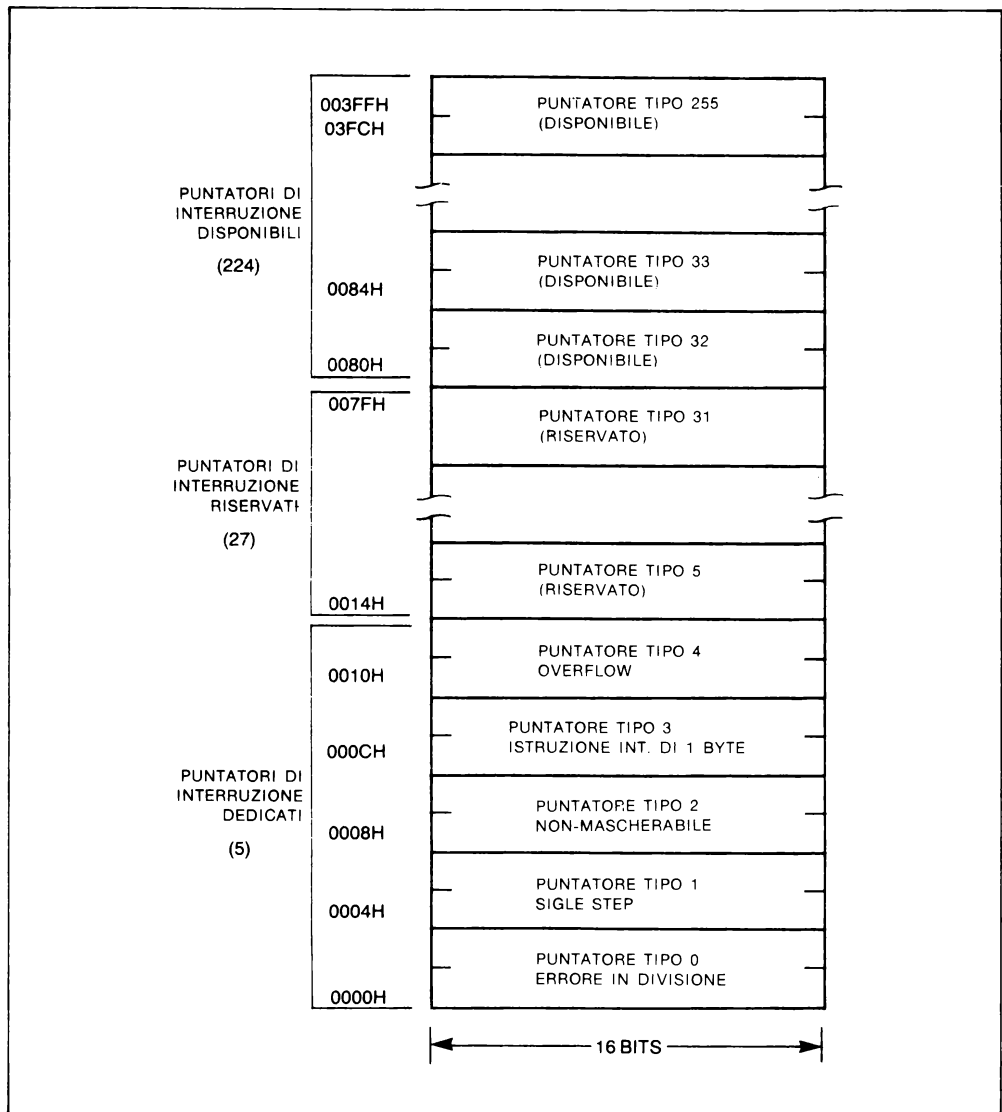



Figura 6.3 — La tabella delle interruzioni per l'8086/8088. Ogni tipo di interruzione richiede 4 byte di memoria.

byte sono necessari per immagazzinare i registri a 16 bit CS e IP utilizzati nella generazione dell'indirizzo a 20 bit della routine di servizio dell'interruzione. I primi 2 dei 4 byte contengono il valore del registro IP. Gli ultimi 2 byte contengono il valore del registro CS, come mostrato in Figura 6.4.

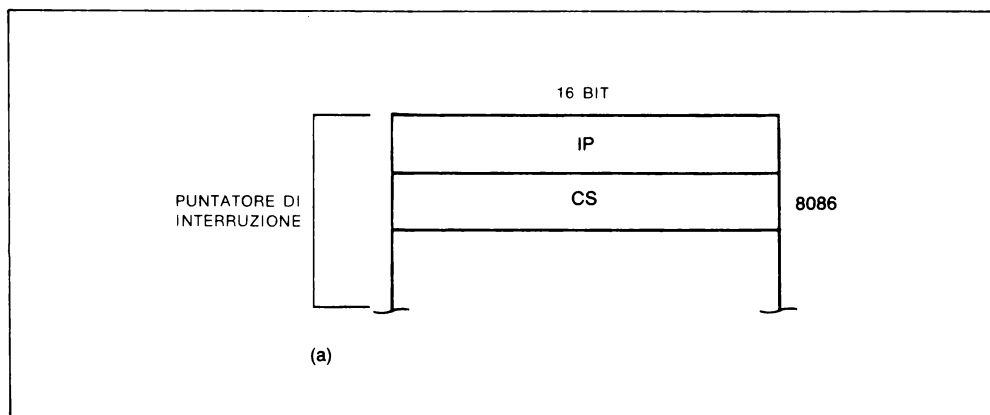



Figura 6.4a — Il puntatore di interruzione per l'8086 richiede due parole di 16 bit. L'indirizzo inferiore è uguale al registro IP; l'indirizzo superiore è uguale al registro CS.

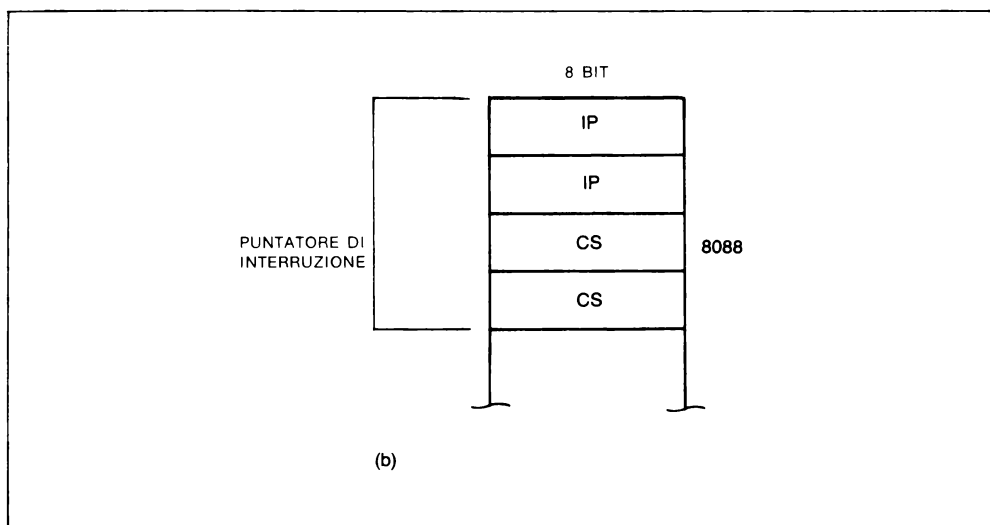



Figura 6.4b — Il puntatore d'interruzione per l'8088 richiede 4 byte contigui di memoria.

Tipi di interruzione riservati

Alcuni dei 255 tipi di interruzioni possibili sono riservati per usi specifici. Quella che segue è una lista dei tipi di interruzioni da 0 a 4 e dei loro usi specifici nel sistema:

TIPO 1 riservato per il single step
TIPO 2 riservato per le interruzioni NMI

TIPO 3 riservato per l'istruzione di 1 byte di interruzione software

TIPO 4 riservato per le interruzioni di overflow con segno

Nota: l'INTEL CORPORATION [che ha progettato l'8086/8088] ha richiesto che i tipi di interruzioni da 5 a 31 [indirizzi di memoria 00014H-0007FH] siano riservati per futuri prodotti Intel. In altre parole, utilizzando un'interruzione di tipo 5-31, il sistema protrebbe non essere capace di usare alcuni prodotti futuri progettati dalla Intel).

INGRESSO INTR

Discutiamo ora l'ingresso INTR. Questo ingresso di interruzioni può essere logicamente abilitato e disabilitato attraverso il software. Quando questo ingresso è disabilitato, la CPU non soddisfa nessuuna richiesta esterna; si può infatti pensare che per la CPU queste richieste non esistono. Nel Capitolo 4 abbiamo discusso l'istruzione CLI (azzeramento del flag di abilitazione delle interruzioni). Questa istruzione software viene usata per disabilitare l'ingresso INTR.

Possiamo abilitare l'ingresso INTR alla CPU 8086/8088 usando l'istruzione STI (che pone uguale a 1 il flag di interruzione). Per esempio, supponiamo che l'ingresso INTR sia abilitato e che sia stata fatta una richiesta esterna. La seguente discussione spiegherà come la CPU tratta questi tipi di richieste.

Segnale di riconoscimento dell'interruzione INTR

Quando l'ingresso INTR viene preso in considerazione dalla CPU, viene generato un segnale esterno, chiamato riconoscimento dell'interruzione (interrupt acknowledge). Questo segnale permette all'hardware del sistema di mettere un valore ad 8 bit, chiamato tipo dell'interruzione, sul bus dati del sistema.

Generazione dell'indirizzo del tipo dell'interruzione

Si tenga presente che ci sono 255 tipi differenti. La CPU legge poi il tipo dell'interruzione dal bus dati e computa l'indirizzo nella tabella delle interruzioni per il tipo in questione. Questo viene fatto moltiplicando per 4 il numero del tipo letto dal bus dati del sistema. Per esempio, supponiamo che l'hardware esterno metta un 2FH sul bus dati del sistema quando ha ricevuto il riconoscimento dell'interruzione. 2F è il numero del tipo.

L'indirizzo del tipo con questo numero nella tabella di interruzione è 4 × 2FH, cioè 000BCH.

Dopo questa operazione, i flag vengono salvati sullo stack del sistema, e le interruzioni vengono disabilitate.

La CPU salva poi registri IP e CS sullo stack del sistema. Dopo questo, la CPU carica i nuovi valori dei registri CS e IP dalle locazioni di memoria 000BCH-000BFH. Quando questi registri sono stati caricati, il microprocessore preleva la successiva istruzione dall'indirizzo a 20 bit generato dai nuovi valori dei registri CS e IP.

Gestione delle interruzioni provenienti dall'ingresso INTR

Rivediamo ora la lista delle operazioni che la CPU compie quando gestisce una richiesta INTR:

1) Viene generato un segnale esterno di riconoscimento dell'interruzione.
2) Viene letto dal bus dati del sistema il codice del tipo da 0 a 255.
3) Il registro FLAG viene salvato nello stack del sistema.
4) Vengono disabilitate le interruzioni e il modo single step.
5) I registri CS e IP vengono salvati nello stack del sistema.
6) Il registro IP viene caricato con i dati prelevati all'indirizzo di memoria (tipo × 4) e (tipo × 4) + 1.
7) Il registro CS viene caricato con i dati prelevati all'indirizzo di memoria (tipo × 4) + 2 e (tipo + 4) + 3.
8) La CPU va a prendere la successiva istruzione all'indirizzo di memoria a 20 bit generato dai nuovi valori dei registri CS e IP appena caricati dalla tabella delle interruzioni.

La Figura 6.5 mostra un diagramma di flusso di ciò che avviene durante il trattamento di un'interruzione con l'8086/8088.

INTERRUZIONI INTERNE

Le interruzioni esterne sono solo uno dei modi per generare le interruzioni con l'8086/8088. Un'altra tecnica è quella di generarle via software. Usando speciali istruzioni software è possibile abilitare la CPU a rispondere ad ogni tipo di interruzione da 0 a 255. Ecco come si fa.

Un'istruzione software che genera un'interruzione è INT (presentata nel Capitolo 4). Ci sono 2 modi di usare questa istruzione particolare, come istruzione ad 1 byte o a 2 byte. Se si usa un'istruzione INT a 1 byte questa è codificata come CCH.

Istruzione INT a 1 byte: interruzione di breakpoint

Quando la CPU esegue questa istruzione a 1 byte, viene automaticamente generata un'interruzione di tipo 3 (breakpoint), esattamente nello stesso modo dell'ingresso INTR. Però, di-

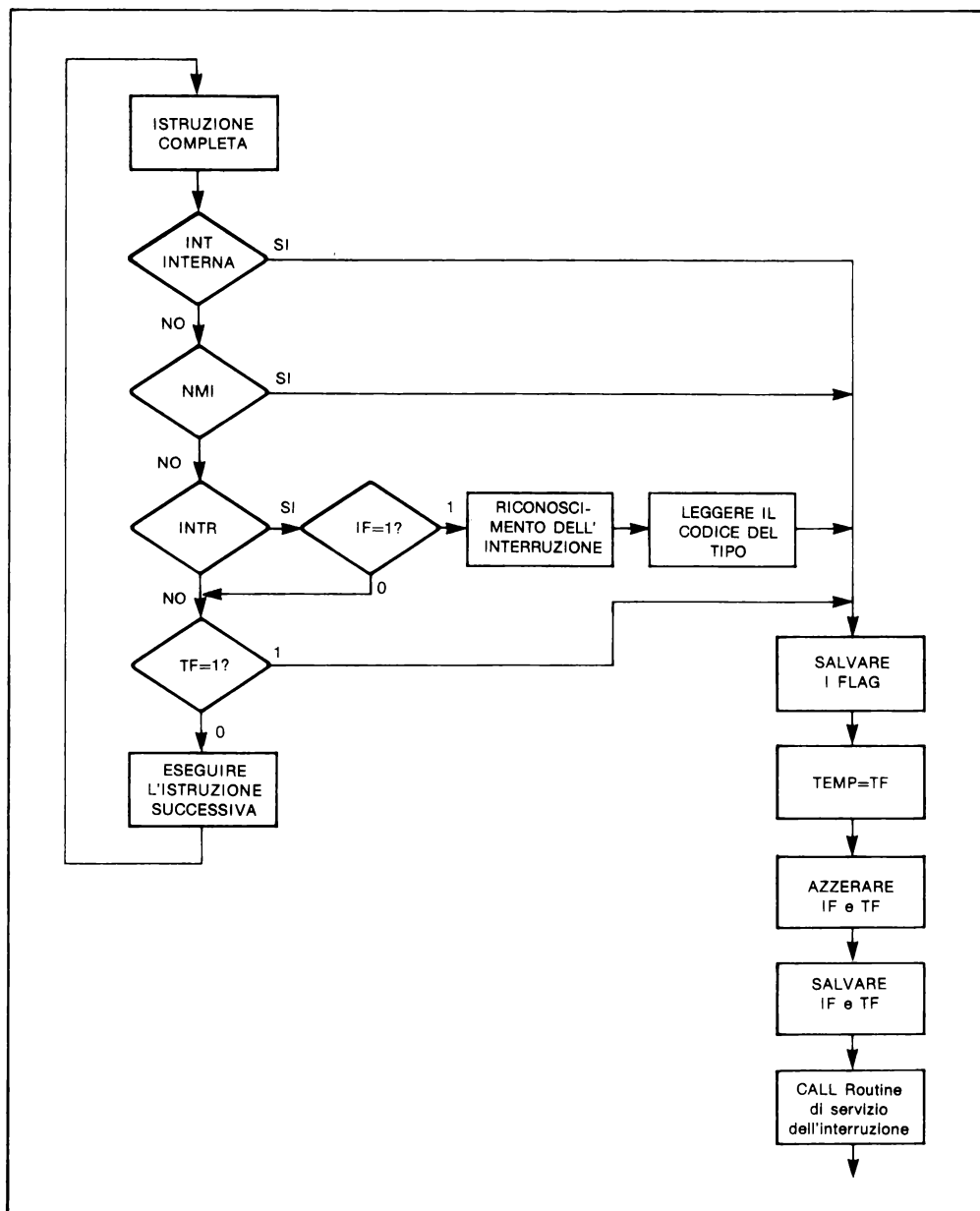



Figura 6.5 — Questo diagramma mostra ciò che succede durante il trattamento delle interruzioni nell'8086/8088.

Versione a 2 byte dell'istruzione INT

versamente dall'ingresso INTR al microprocessore, l'interruzione software non può essere disabilitata o mascherata.

Supponiamo ora di usare la versione a 2 byte dell'istruzione

INT. Il primo byte sarà codificato come CDH.

Il secondo byte sarà uguale ad un numero ad 8 bit compreso fra 0 e 255 (inclusi). Questo numero definisce il tipo di interruzione che si vuole generare. Per esempio, supponiamo che si voglia generare un'interruzione di tipo 96H.

L'istruzione INT di 2 byte verrà codificata come CD96H.

L'indirizzo esadecimale nella tabella delle interruzioni per l'IP sarà uguale a 4 × 96H = 258 in esadecimale. Gli indirizzi 258H e 259H conterranno il valore del registro IP per la routine di servizio dell'interruzione. Gli indirizzi 25AH e 25BH conterranno il valore del registro CS per la routine di servizio dell'interruzione.

Interruzione su overflow

C'è un'altra istruzione software che può causare la generazione automatica di un'interruzione del tipo 4: l'istruzione INTO. Quando il flag di overflow (OF) vale 1 viene eseguita questa istruzione, si genera un'interruzione interna. Fare riferimento al Capitolo 1 per una completa discussione sul significato dell'overflow.

Ritorno da un'interruzione

Ora che abbiamo imparato come la CPU inizia l'esecuzione di una routine di servizio di un'interruzione, discutiamo come la CPU fa ritorno dalla routine di servizio al programma principale. Quando la CPU inizia la routine di servizio, i registri FLAG, CS e IP vengono salvati. Queste sono essenzialmente le informazioni di cui ha bisogno la CPU per ritornare all'indirizzo che conteneva prima dell'interruzione.

Durante l'esecuzione di una routine di interruzione, possono essere cambiati i valori di qualche registro interno. Se è importante salvare i valori originali di questi registri, è opportuno salvarli nello stack. La Figura 6.6 mostra un tipico inizio di una routine di servizio di una interruzione.

Un altro punto da notare circa l'inizio di una routine di servizio di una interruzione è che tutte le interruzioni sono state disabilitate. Se si vuole che altre interruzioni vengano prese in considerazione dalla CPU mentre sta trattando questa interruzione, l'istruzione che pone uguale ad 1 il flag di abilitazione delle interruzioni (STI) dovrà essere una delle prime istruzioni che viene eseguita nella routine di servizio dell'interruzione.

Questo è dimostrato dal programma parziale nella Figura 6.7.

```
;
; INIZIO DELLA ROUTINE DI SERVIZIO DELL'INTERRUZIONE
;
        PUSH AX         ;SALVA IL REGISTRO AX
        PUSH BX         ;SALVA IL REGISTRO BX
        PUSH CX         ;SALVA IL REGISTRO CX
        PUSH DX         ;SALVA IL REGISTRO DX
;
; ORA INIZIA IL CODICE REALE DELLA ROUTINE DI INTERRUZIONE
;
  MOV AX,NUM
```

Figura 6.6 — Il tipico inizio di una routine di servizio di un'interruzione che mostra come i registri importanti vengano salvati sullo stack.

Uso dell'istruzione RETI per ritornare da una routine di interruzione

Supponiamo ora di aver terminato il trattamento dell'interruzione. Come si ritorna al programma principale? Si usa l'istruzione RETI. Quando questa istruzione viene eseguita, i registri FLAG, IP e CS sono prelevati dallo stack. Prima di ritornare al programma principale è importante assicurarsi che le interruzioni siano ancora abilitate (se ciò è desiderato). Inoltre sarà necessario prelevare ogni registro interno che era stato immesso nello stack durante la routine di servizio dell'interruzione. La Figura 6.8 mostra un esempio di routine di servizio di una interruzione che dimostra tutti questi punti.

```
;
; INIZIO DELLA ROUTINE DI SERVIZIO DELL'INTERRUZIONE
;
        STI             ;ABILITA NUOVAMENTE LE INTERRUZIONI
        PUSH AX         ;SALVA IL REGISTRO AX
        PUSH BX         ;SALVA IL REGISTRO BX
        PUSH CX         ;SALVA IL REGISTRO CX
        PUSH DX         ;SALVA IL REGISTRO DX
;
; ORA INIZIA IL CODICE REALE DELLA ROUTINE DI INTERRUZIONE
;
  MOV AX,NUM
```

Figura 6.7 — Questo diagramma mostra l'inizio di una routine di interruzione che permette ulteriori interruzioni. Ciò viene reso possibile, all'inizio della routine di servizio, dall'istruzione STI.

```
;
; INIZIO DELLA ROUTINE DI SERVIZIO DELL'INTERRUZIONE
;
        PUSH AX         ;SALVA IL REGISTRO AX
        PUSH BX         ;SALVA IL REGISTRO BX
        PUSH CX         ;SALVA IL REGISTRO CX
        PUSH DX         ;SALVA IL REGISTRO DX
;
; ORA INIZIA IL CODICE REALE DELLA ROUTINE DI INTERRUZIONE
;
        MOV AX,NUM
        MUL AX,BX
;
; UN'ALTRA PARTE DELLA ROUTINE VA QUI
;
; FINE DELLA ROUTINE
;
        POP DX
        POP CX
        POP BX
        POP AX
        STI             ;ABILITA LE INTERRUZIONI
        RETI            ;RITORNO DALL'INTERRUZIONE
```

Figura 6.8 — Un esempio di routine di interruzione per l'8086/8088.

RESET DELL'8086/8088

Nei precedenti esempi abbiamo mostrato come le interruzioni NMI, INTR e SOFTWARE operano per la CPU. Discutiamo ora un altro tipo di interruzione: l'ingresso RESET sull'8086/8088. Quando la linea di ingresso RESET viene attivata, la CPU comincia l'esecuzione in una ed una sola maniera. Questo permette una partenza corretta, una ripartenza o un "power-up" del sistema.

In seguito alla richiesta di RESET i registri della CPU sono inizializzati nel seguente modo:

FLAG	= azzerato
PUNTATORE ALL'ISTRUZIONE	= 0000H
REGISTRO CS	= FFFFH
REGISTRO DS	= 0000H
REGISTRO SS	= 0000H
REGISTRO ES	= 0000H

Basandosi su queste definizioni dei registri, la prima istru-

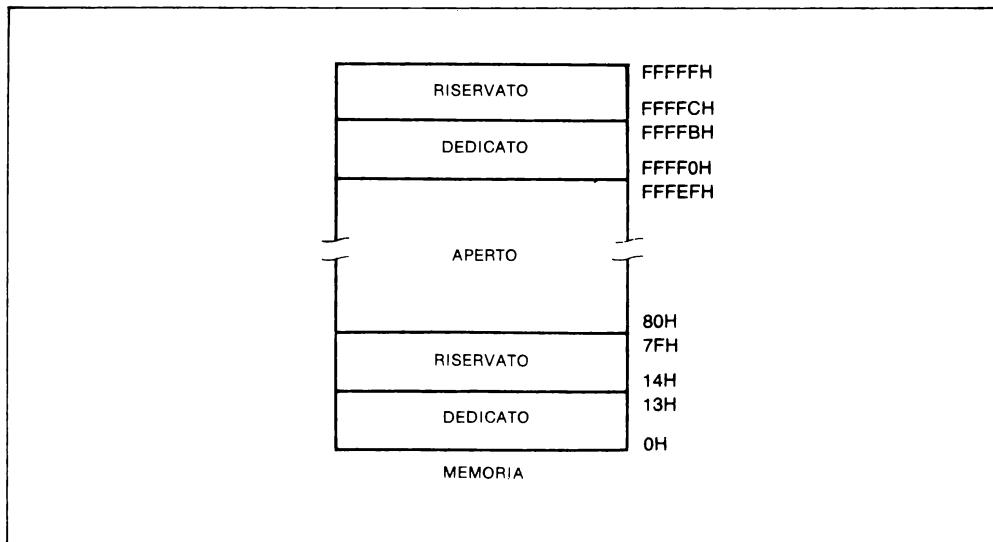



Figura 6.9 — Una mappa della memoria che mostra le locazioni riservate dell'8086/8088.

zione verrà presa dalla seguente locazione di memoria:

$$(CS \times 16) + IP = FFFF0H + 0000H = FFFF0H$$

La prima locazione dopo il reset è la locazione assoluta FFFF0H nella parte alta della memoria. Ricordiamo che il produttore raccomanda di non usare queste locazioni in cima alla memoria, da FFFF0H a FFFFFH, eccetto che per gli scopi per cui sono state intese. Le locazioni di memoria riservate per l'8086/8088 sono mostrate in Figura 6.9.

SOMMARIO

In questo capitolo abbiamo esaminato le interruzioni per l'8086/8088. La nostra discussione è iniziata con un'introduzione generale di una interruzione. Da qui si è continuato parlando dell'interruzione INTR mascherabile e della NMI non mascherabile. In ciascun caso abbiamo spiegato come le richieste di interruzione sono gestite dalla CPU.

Abbiamo poi discusso come la CPU genera le interruzioni interne usando le istruzioni INT e INTO. Abbiamo terminato il capitolo mostrando come l'8086/8088 risponde internamente al RESET.

Una volta che si sono comprese le informazioni di questo capitolo si è in grado di capire meglio come la CPU può essere programmata per trattare differenti richieste di interruzioni da parte del sistema hardware periferico.

CAPITOLO 7

INPUT E OUTPUT PER L'8086/8088

INTRODUZIONE

Un sistema 8086/8088 consiste normalmente di uno o più dispositivi di input e/o output. In questo capitolo discuteremo l'architettura di input/output (I/O) della CPU 8086/8088. Inoltre presenteremo speciali istruzioni per l'I/O e daremo una spiegazione di come possano venir usate. Daremo parecchi esempi che illustrano come uno 8086/8088 interfaccia con differenti dispositivi di I/O e li controlla.

CHE COSA SONO L'INPUT E L'OUTPUT?

Nel Capitolo 2 abbiamo esaminato l'archittetura generale di un sistema basato su microprocessore. La Figura 7.1 ne fornisce un rapido riassunto. In questa figura vediamo che il microprocessore comunica con ROM, RAM, e I/O. ROM e RAM sono messe insieme per formare la memoria del sistema (input/output). Con l'8086/8088 la memoria del sistema ha indirizzi validi da 00000H a FFFFFH. È da notare che il blocco I/O non è incluso in questo spazio di memoria.

I/O "mappato in memoria" e "a mappa di I/O"

Alcuni microprocessori lasciano dello spazio disponibile nella memoria per l'I/O. Questi microprocessori includono il 6800, il 6502, il 6809 ed il 68000 solo per citarne alcuni.

Nel caso dei microprocessori che utilizzano lo spazio della memoria per l'I/O si dice che essi usano un "I/O mappato in memoria".

L'8086 e l'8088 non usano l'I/O mappato in memoria. Perciò, tutto lo spazio nella memoria del sistema può essere usato per per la memoria, dato che il sistema I/O ha il proprio spazio di indirizzamento. L'architettura di I/O di questo tipo viene chiamata "I/O a mappa di I/O".

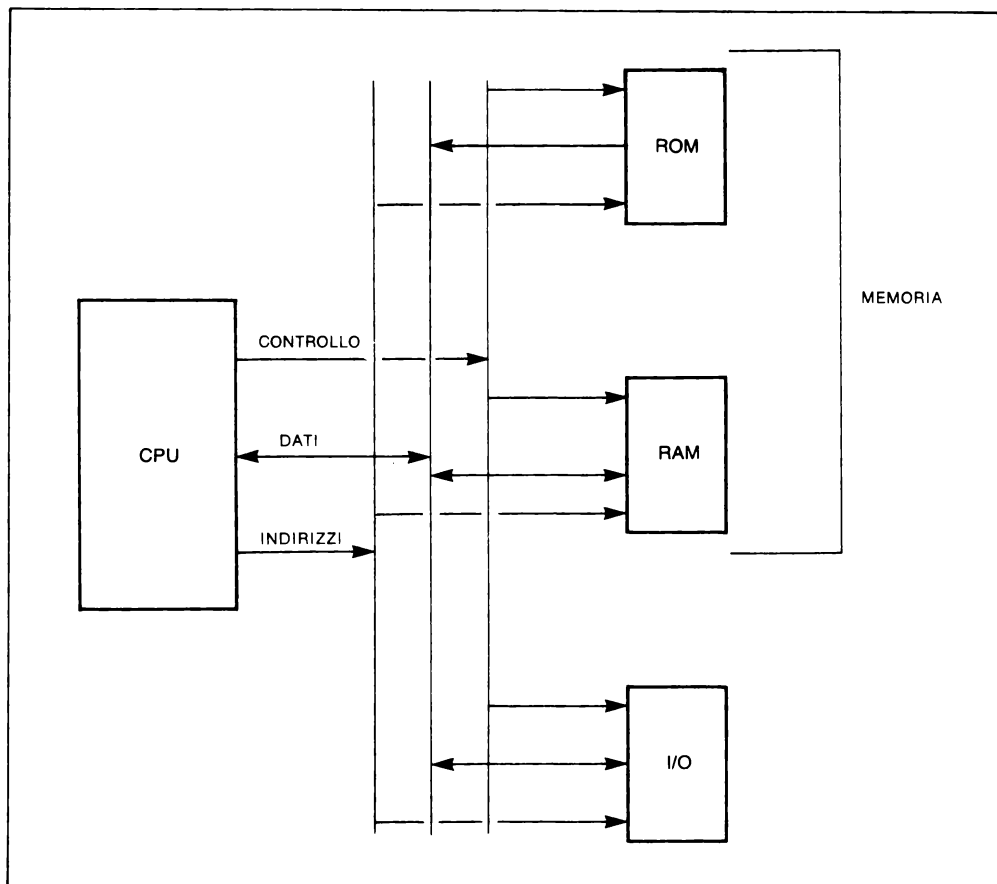



Figura 7.1 — Il diagramma mostra l'architettura generale a 3 bus di un sistema microprocessore.

Definizione di un'operazione di Input/Output

Un'operazione di I/O può essere definita come segue:

Input: Quando il microprocessore legge i dati da una fonte di dati che non è la memoria del sistema.

Output: Quando il microprocessore scrive dei dati ad una destinazione che non è la memoria del sistema.

Separazione delle linee di controllo

Abbiamo già menzionato in precedenza in questo testo che il bus di controllo del sistema definisce il tipo di comunicazione che ha luogo. In altre parole, se il sistema usa "I/O a mappa di I/O, ci sono linee di controllo separate per il sistema di I/O e per il sistema di memoria. Per esempio, il sistema di memoria

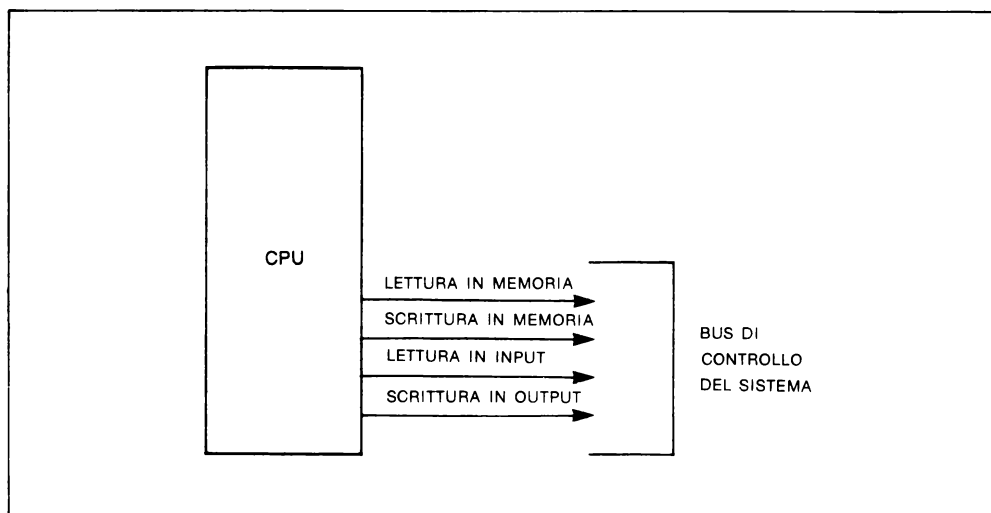



Figura 7.2 — I segnali di controllo del sistema per la memoria sono "lettura in memoria" e "scrittura in memoria". I segnali di controllo corrispondenti per l'I/O sono "lettura in input" e "scrittura in output".

usa linee di controllo etichettate "lettura in memoria" e "scrittura in memoria", mentre il sistema di I/O usa "lettura in input" e "scrittura in output"; come mostra la Figura 7.2.

INDIRIZZAMENTO DI I/O

Porte di Input/Output

In un tipico sistema microprocessore ci sono abitualmente parecchie porte di I/O. (N.B.: Un porta è un posto univocamente identificato, diverso dalla memoria del sistema, per leggere o scrivere dati.

Infatti, una porta è simile ad un'unica locazione nella memoria del sistema da cui i dati verranno letti o in cui verranno scritti durante un'operazione in memoria). Ogni porta nel sistema I/O ha un indirizzo particolare, chiamato codice di selezione della porta. (Vedere Figura 7.3). Per generare il codice di selezione della porta, le linee di indirizzo del sistema A0-A15 vengono decodificate con logica per rispondere ad una combinazione specifica. Per esempio, una porta può avere un codice di selezione della porta di 0057H, come è mostrato nella Figura 7.4.

Porte indirizzabili con gli 8086/8088

Per gli 8086/8088 ci sono solo 16 linee di indirizzo sulle 20 (totali) linee di indirizzo del microprocessore disponibili per indirizzare l'I/O. Questo dà un totale di 65.535 porte di input e

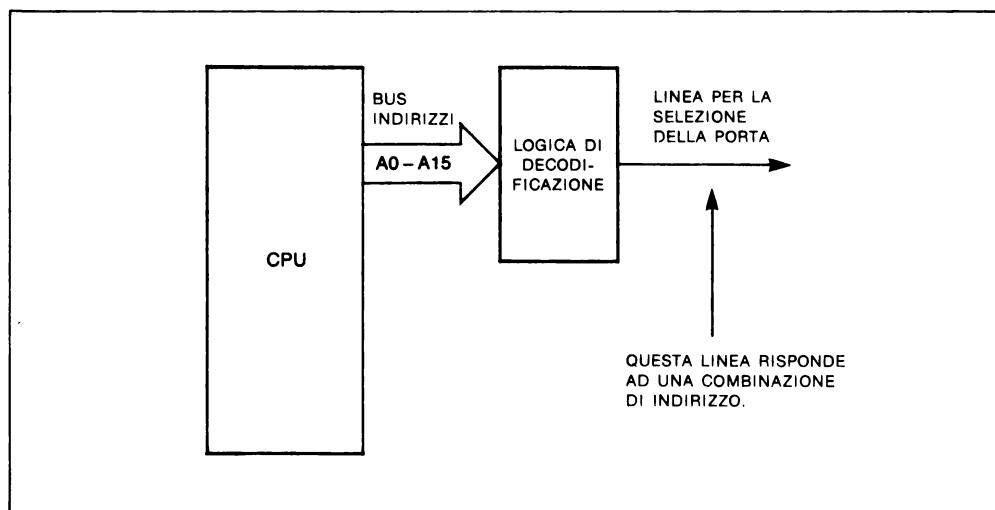



Figura 7.3 — Ciascuna porta I/O del sistema è progettata per rispondere elettricamente ad un'unica combinazione sul bus indirizzi del sistema.

di output disponibili nel sistema. La Figura 7.5 mostra una mappa di memoria dello spazio disponibile per l'I/O con gli 8086/8088.

Benchè il sistema di I/O ed il sistema di memoria siano

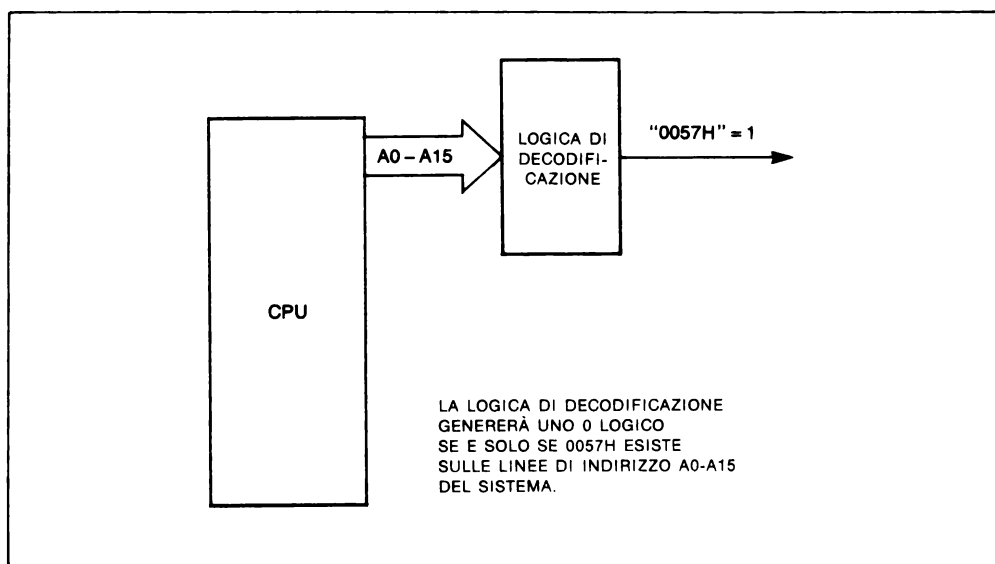



Figura 7.4 — Questa porta I/O risponderà elettricamente all'indirizzo 0057H del sistema.

completamente separati, questi due sistemi usano le stesse linee di indirizzo. Ciò significa che si puo' avere un indirizzo della memoria di 00F4H ed un indirizzo di I/O di 00F4H. Si può fare una distinzione fra questi due sistemi analizzando le linee del bus di controllo del sistema: durante un'operazione di memoria è attiva la linea di controllo della lettura in memoria o quella della scrittura in memoria durante un'operazione I/O è attiva la linea della lettura dell'input o della scrittura dell'output.

Spazio di I/O riservato

La Figura 7.5 mostra un'area dello spazio di I/O - indirizzi tra 00F8H e 00FFH - che è etichettata "riservato".

Quest'area è etichettata "riservato" perchè l'Intel Corporation ha richiesto che non venga usata dalle vostre applicazioni, poichè verrà usata da futuri prodotti Intel.

Cos'è un dispositivo di I/O?

Un dispositivo I/O può essere definito come qualsiasi componente hardware controllato dal sistema. Tale dispositivo può avere una o più porte di I/O o indirizzi di I/O associati ad

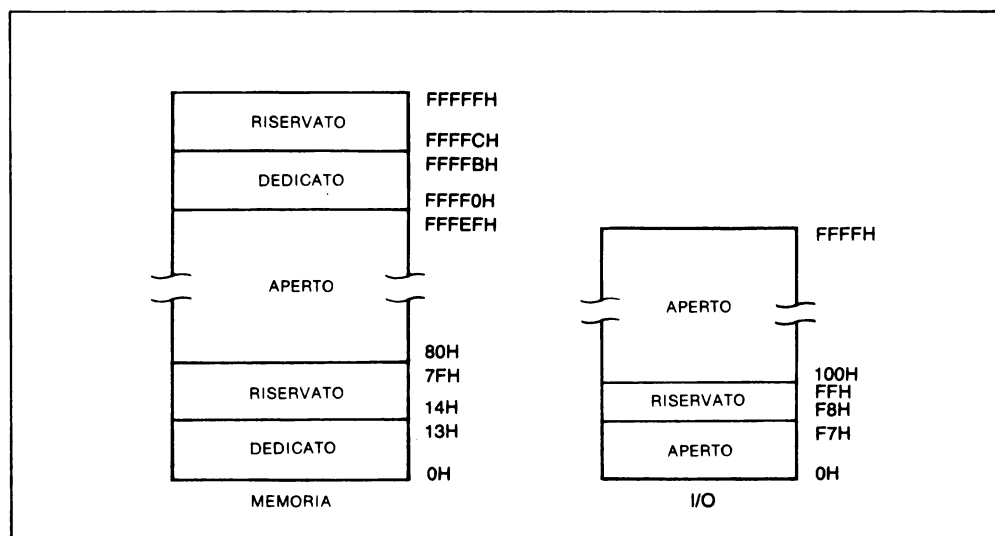



Figura 7.5 — Una mappa della memoria che mostra lo spazio disponibile per la memoria di sistema e l'I/O.

esso, come è mostrato nella Figura 7.6. Esempi di dispositivi di I/O sono i chips LSI (integrazione a larga scala), come un controllore di floppy disk o un timer.

L'ISTRUZIONE DI INPUT

L'istruzione IN viene usata per leggere i dati da una porta di input su un microprocessore. Quest'istruzione è scritta così:

IN accumulatore, porta

Registri utilizzati nell'istruzione IN

L'istruzione IN trasferisce un byte o una parola di dati da un indirizzo di una porta di input all'accumulatore. Se trasferisce un byte di dati, questo viene immagazzinato nel registro AL; se è una parola, viene messa nel registro AX.

Indirizzamento di una porta fissa

Codifica dell'istruzione IN per una porta fissa

Ci sono due modi per utilizzare l'istruzione IN: o con una porta fissa o con una porta variabile. Quando viene usata un'istruzione IN con una porta fissa, essa viene codificata come è mostrato nella Figura 7.7. Possiamo vedere in questa figura

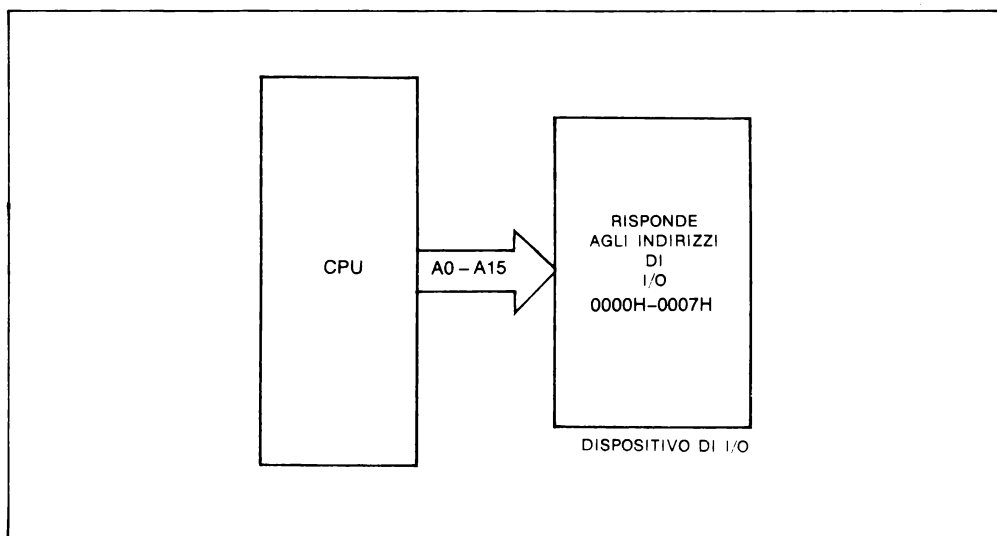



Figura 7.6 — Un dispositivo di I/O risponderà elettricamente ad uno o più indirizzi di porta I/O. Questo dispositivo risponderà a tutti gli indirizzi di I/O da 0000H a 0007H compresi.

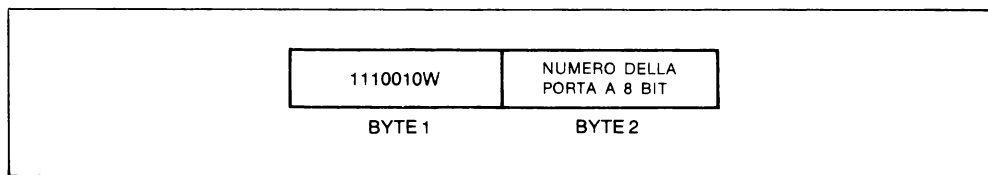



Figura 7.7 — La codifica di un'istruzione INput per una porta fissa.

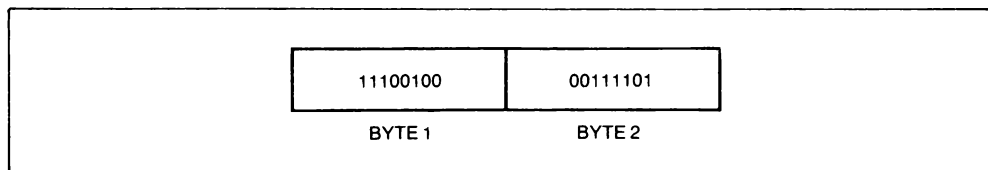



Figura 7.8 — La codifica dell'istruzione IN per una porta fissa: IN AL,3DH.

che il primo byte ha il bit W posto uguale ad un 1 o ad uno 0 logico. Se W è posto ad un 1 logico, il trasferimento è di una parola (16 bit) ed i dati finiscono nel registro AX. Se è posto ad uno 0 logico, il trasferimento è di un byte ed i dati vanno nel registro AL.

Il secondo byte mostrato nella Figura 7.7 è un indirizzo di porta a 8 bit. Dato che ci sono solo 8 bit, usando questa istruzione è possibile accedere ad un totale di 225 porte di I/O uniche. Come esempio di uso di quest'istruzione supponiamo di leggere un dato di un byte alla porta 3DH. La codifica dell'istruzione sarebbe quella mostrata in Figura 7.8. Il modo in cui specificare un trasferimento di byte o di parola è questo:

```
IN AX,porta(parola)
IN AL,porta(byte)
```

Indirizzamento di una porta variabile

Un altro modo in cui l'istruzione IN può essere usata è con una porta variabile. Usando una codifica di porta variabile si ha accesso a tutti i 16 bit del bus indirizzi del sistema per l'indirizzamento di I/O. La codifica del formato con porta variabile dell'istruzione IN è mostrata nella Figura 7.9.

Codifica dell'istruzione IN per una porta variabile

Guardando la Figura 7.9, vediamo che è richiesto solo un byte per usare quest'istruzione. Il bit W è uguale a 1 se si richiede un dato di una parola; è uguale a 0 se si vuole un dato di un byte. L'indirizzo per la porta è contenuto nel registro DX quando l'istruzione viene eseguita. L'usare il registro DX dà 16 bit di indirizzo che possono essere modificati sotto il con-

```
1110110W
```

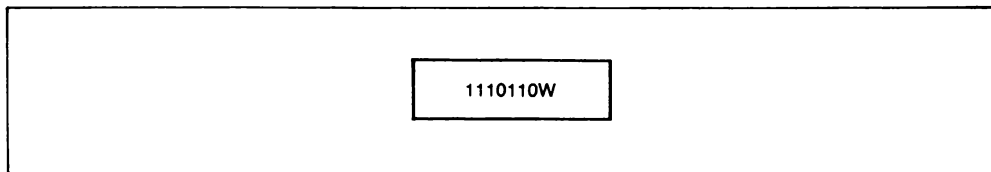



Figura 7.9 — La codifica di un'istruzione INput per una porta variabile. L'indirizzo sarà contenuto nel registro DX della CPU.

trollo del programma. Per esempio, possiamo usare il formato con porta variabile dell'istruzione IN per leggere un dato di una parola dalla porta 05ACH. Il programma mostrato nella Figura 7.10 raggiungerà questo scopo.

L'ISTRUZIONE DI OUTPUT

L'istruzione OUT viene usata per trasferire i dati dalla CPU ad una porta di output. Questa istruzione ha un formato simile all'istruzione IN. Il suo formato è:

OUT porta,accumulatore

Come nell'istruzione INput, anche l'istruzione OUTput può essere usata nel formato con porta variabile o in quello con porta fissa. La Figura 7.11 mostra esempi di due piccoli programmi che usano l'istruzione OUT nei formati con porta fissa e con porta variabile.

```
                1 ;
                2 ;  PROGRAMMA PER USARE LA PORTA VARIABILE
                3 ;  MODO PER L'ISTRUZIONE IN
                4 ;  NUMERO A 16 BIT DELLA PORTA=05AC
                5 ;
0000 BAAC05     6          MOV DX,#05ACH  ;numero della porta nel REG.DX
0003 ED         7          IN AX,DX       ;parola in input da DX a AX
                8 ;
```

Figura 7.10 — Un programma 8086/8088 che mostra l'uso dell'istruzione IN con una porta variabile.

```
               1 ;
               2 ;
0000 BAAC05    3          MOV DX,#05ACH      ;CARICA L'INDIRIZZO NEL REG. DX
0003 EC        4          IN AL,DX           ;INPUT DI UN BYTE DALL'IND.IN DX AD AL
               5 ;
               6 ;    ;OUTPUT DI UN BYTE USANDO IL FORMATO CON PORTA VARIABILE
               7 ;
0004 BA8704    8          MOV DX,#0487H      ;carica l'indirizzo nel REG.DX
0007 EE        9          OUT DX,AL    ;output del dato alla porta variabile
              10 ;
              11 ;
0008 E607     12          OUT 07H,AL         ;output di un byte alla porta fissa=07
              13 ;
```

Figura 7.11 — Questo programma 8086/8088 mostra come vengono usate le istruzioni OUTPUT con una porta fissa e con una variabile.

PORTE DI INPUT E DI OUTPUT DELL'8086

Connessione di una porta in base al tipo di indirizzo

Quando si usa l'8086 bisogna ricordare che ha un bus dati a 16 bit. Ciò significa che gli indirizzi delle porte devono essere coerenti con il byte del dato (cioè, alto o basso). Quando una porta ad un byte ha un indirizzo pari, deve essere fisicamente connessa con il byte più basso (D0-D7) dei 16 bit del bus dati. Una porta ad un byte ad indirizzo dispari deve essere connessa fisicamente con il byte più alto (D8-D15) del bus dati del sistema. L'indirizzo di una porta di una parola dovrebbe essere specificato come un indirizzo pari, come mostrato nella Figura 7.12.

PORTE DI INPUT E DI OUTPUT DELL'8088

Anche utilizzando una CPU 8088, che ha un bus dati di 8 bit, è possibile usare sia le porte di output ad una parola che quelle ad un byte. Le stesse convenzioni che abbiamo discusso per l'8086 possono essere seguite per l'8088. Tuttavia, non c'è bisogno di preoccuparsi per le porte ad un byte indirizzate ad indirizzi pari o dispari, poichè c'è solo un modo per connettere un bus dati di 8 bit.

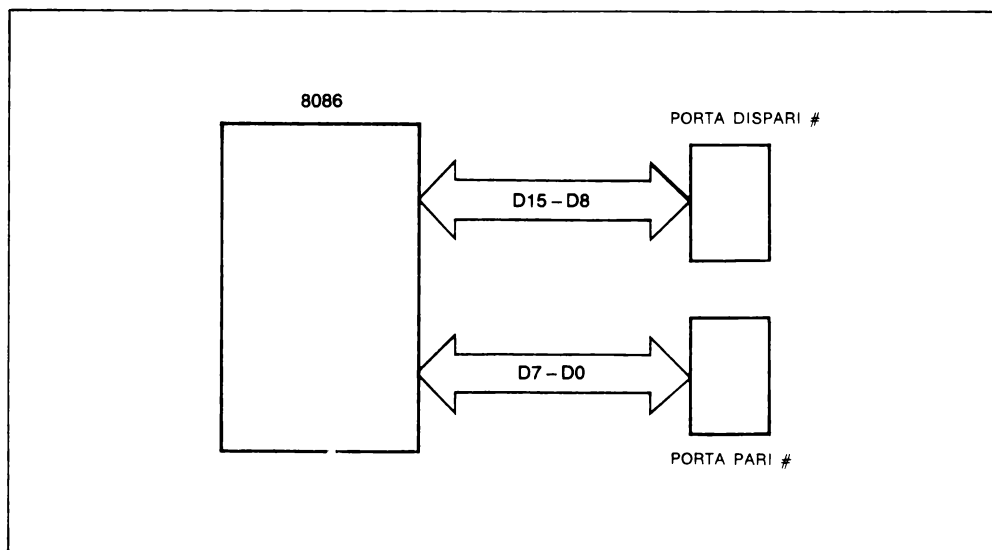



Figura 7.12 — Il diagramma indica che su un sistema 8086, una porta ad 8 bit ad un indirizzo dispari deve essere collegata alle linee dati superiori (D8-D15), e che una porta ad 8 bit ad un indirizzo pari deve essere collegata alle linee dati inferiori (D0-D7).

Se state usando una porta di output ad una parole, il byte più basso della porta dovrebbe esseread un indirizzo pari ed il byte più alto ad un indirizzo dispari. Ciò è coerente con l'orga-

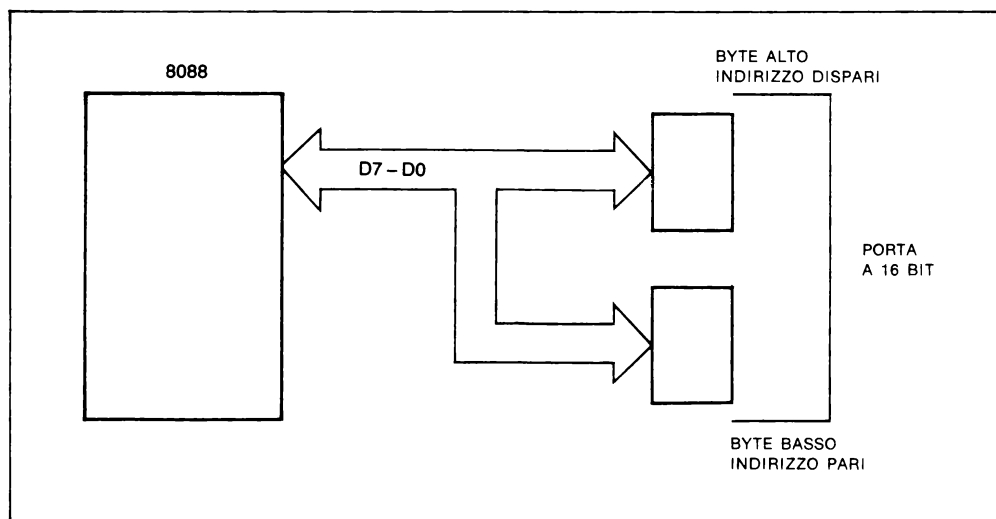



Figura 7.13 — Le porte di I/0 in un 8088 possono avere indirizzi pari o dispari. Se una porta a 16 bit viene specificata, il byte basso dovrebbe essere ad un indirizzo pari, ed il byte alto ad un indirizzo dispari.

nizzazione della memoria di sistema dell'8088, come si vede nella Figura 7.13.

Generazione di un segnale con il software

Applichiamo ora le informazioni imparate sulle istruzioni dell'8086/8088 e sull'I/O a vari problemi generali e alle interfacce usate nei sistemi di microcomputer. Cominceremo col generare un segnale che accenderà o spegnerà alcune parti dell'hardware, come un relè.

Per generare un segnale, il calcolatore deve accendere o spegnere un dispositivo di output. Per raggiungere questo scopo, il calcolatore deve trasformare il livello del voltaggio elettrico nel dispositivo da uno 0 logico ad un 1 logico, o da ùn 1 logico ad uno 0 logico. Per esempio, supponiamo che un relè esterno sia connesso al bit 0 di una porta di output ad 8 bit che chiameremo OUT1. (OUT1 è il nome variabile che rappresenta l'indirizzo [8 o 16 bit] della porta di output).

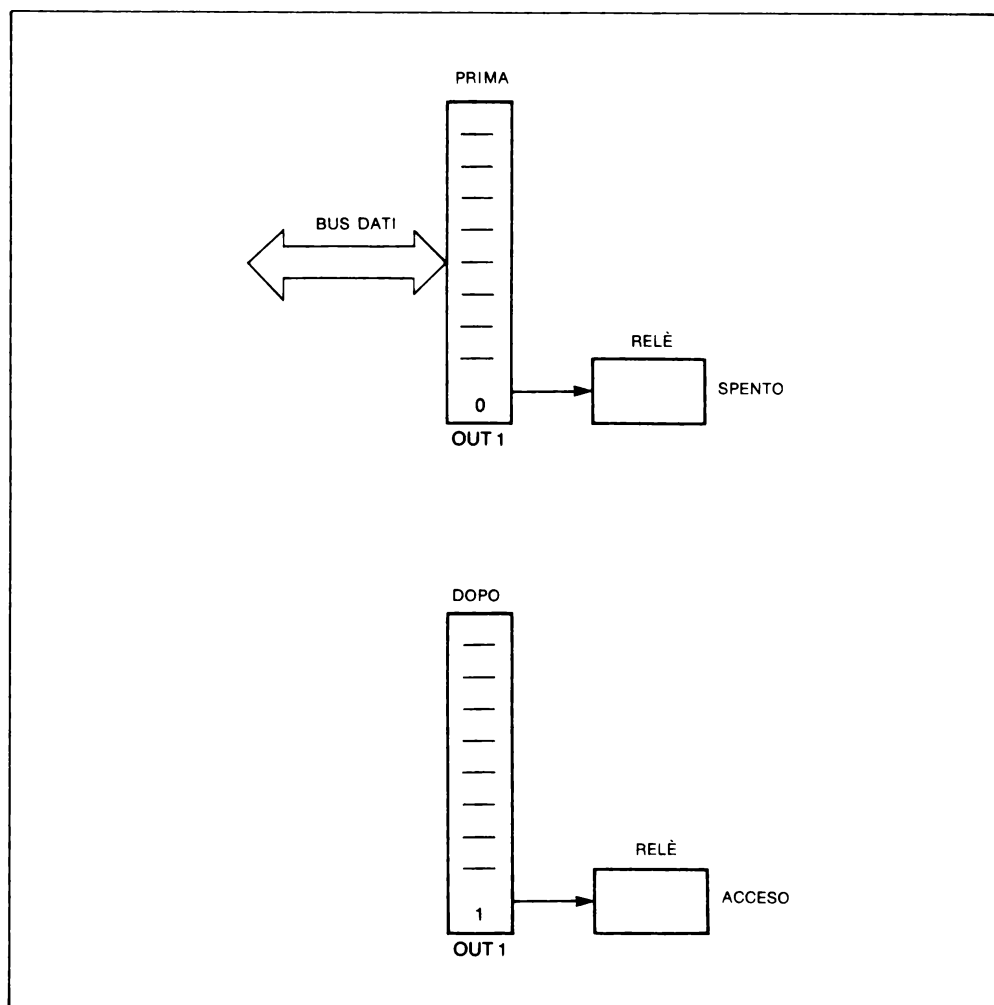
Per accendere il relè si deve scrivere un 1 logico al bit 0 di OUT1. Per spegnere il relè si deve scrivere uno 0 logico. Ecco il programma che accenderà il relè:

```
MOV AL,00000001B     B = BINARIO
OUT OUT1,AL          OUTPUT DI 1 NEL BIT 0 DELLA
                     PORTA OUT1
```

In questo esempio abbiamo presunto che le posizioni degli altri bit della porta di output siano trascurabili.

Tuttavia, in una situazione normale, questo potrebbe non essere il caso; essi potrebbero essere collegati con altri relè nel sistema. In questo caso, non vorremmo cambiare lo stato logico di questi bit. Possiamo assicurare che ciò non accadrà inserendo un'altra istruzione nel programma:

```
IN  AL,OUT1          LEGGE IL VALORE DELLA PORTA
                     DI OUTPUT
OR  AL,00000001B     PONE A 1 SOLO IL BIT 0, GLI ALTRI
                     BIT RIMANGONO INVARIATI
OUT OUT1,AL          OUTPUT ALLA PORTA OUT1
```

Questo programma presume che noi possiamo leggere i contenuti della porta di output OUT1 o che i contenuti siano stati messi in una locazione di memoria a cui noi abbiamo accesso. La Figura 7.14 mostra un diagramma a blocchi del concetto accensione e spegnimento di un relè.

Fiura 7.14 — Questo diagramma mostra come la linea di un segnale di una porta output può essere usata per accendere e spegnere un relè.

Generazione di un impulso

Possiamo generare un impulso allo stesso modo in cui possiamo accendere o spegnere un relè. Prima dobbiamo porre uguale ad un 1 logico un bit in una porta di output e poi cambiarlo ad uno 0 logico dopo un certo periodo (finito) di tempo (Vedere Figura 7.15). Quando l'impulso è generato, dobbiamo seguire la sua ampiezza con il software. Per far ciò, dobbiamo generare un ritardo calcolato.

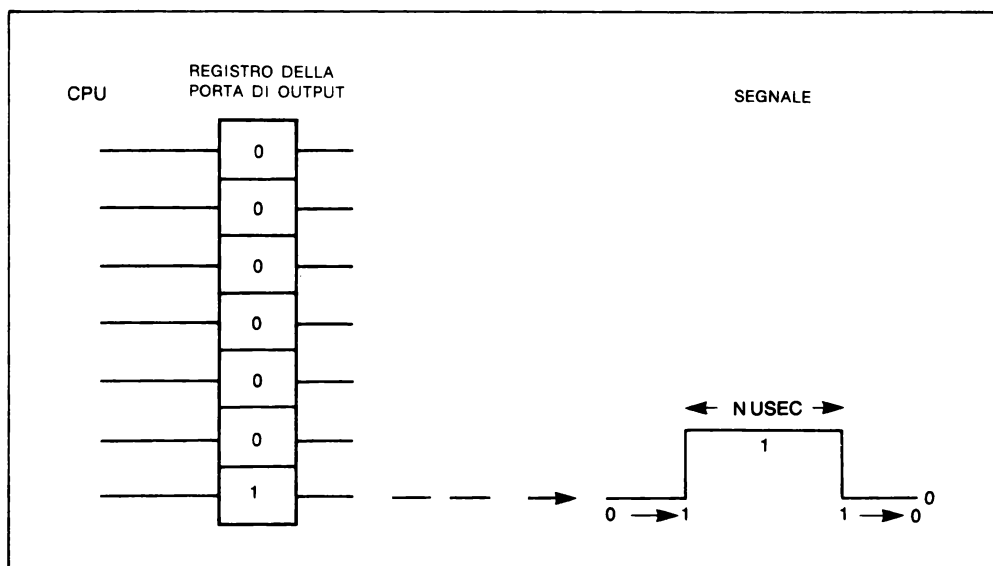



Figura 7.15 — Un impulso può essere generato ponendo ad alto un bit ad una porta output per una lunghezza di tempo fissata, riportandolo poi ad uno 0 logico.

Generazione di un ritardo

I ritardi possono essere generati usando sia il software che l'hardware. Per ora, usiamo il software. In un altro esempio, genereremo un ritardo con l'hardware usando un timer programmabile. I ritardi programmati si ottengono con un conteggio. Per contare, un registro contatore viene dapprima caricato con un valore, poi viene decrementato. Il programma cicla su se stesso e continua a decrementare il contatore finchè questo raggiunge il valore 0. La quantità totale del tempo usato da questo processo realizza il ritardo richiesto.

Per generare un ritardo specifico, dobbiamo sapere quanto tempo ci vuole per eseguire ogni istruzione data una certa frequenza di clock. L'esecuzione di un'istruzione richiede un numero fisso di cicli di clock, ed è questo numero che deve essere trasformato in tempo, basandosi sul periodo del clock del sistema. (Nota: Per questo esempio, non ci preoccuperemo dei tempi esatti).

Generiamo ora un ritardo per la lunghezza di tempo richiesta per decrementare il registro AX nel ciclo. Ecco il programma:

```
        MOV AX,0F45H  CARICA AX CON IL
                      NUMERO TOTALE DEI CICLI
BACK    DEC AX        AX = AX - 1
        JNZ BACK      SE NON È ZERO
                      DECREMENTA ANCORA
```

La prima linea di istruzioni carica il registro AX con il valore immediato di F45H. La linea seguente, etichettata con "BACK", decrementa il registro AX di 1. Dopo di che, l'istruzione JNZ (saltare se non uguale a zero) inizia un salto alla linea di istruzione etichettata BACK se il risultato del decremento non è uguale a zero. Se il valore del registro AX è uguale a 0, il programma esegue l'istruzione che segue l'istruzione JNZ. Un diagramma di flusso di questo tipo di ritardo appare nella Figura 7.16.

La Figura 7.17 presenta un programma per generare un impulso al bit 0 della porta di output OUT1. È da notare che questo programma usa il programma di ritardo che abbiamo appena descritto.

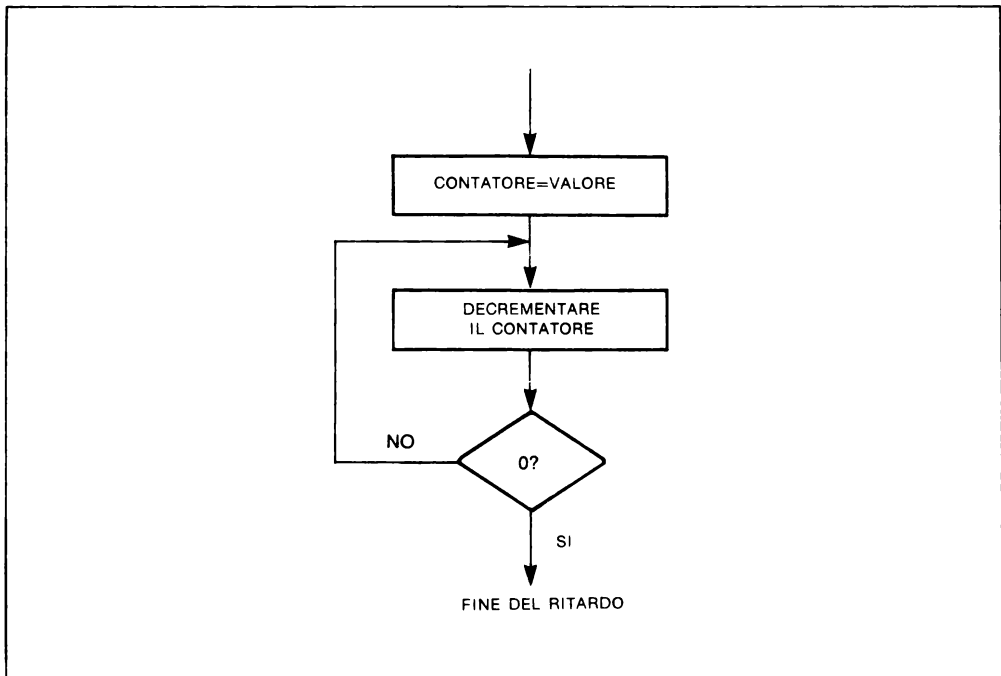



Figura 7.16 — Il diagramma mostra come può essere generato un ritardo software.

```
                1 ;
                2 ;   PROGRAMMA PER GENERARE UN RITARDO
                3 ;   VIENE USATO AX
                4 ;
                5 ;   PORTA DI OUTPUT OUT1=58H
                6 ;   BIT 0 È IL BIT DI CONTROLLO
                7 ;
                8 OUT1     EQU 58H
                9 ;
0000 B001      10  MOV AL,#01 ;PONE A 1 IL BIT DI CONTROLLO
0002 E658      11  OUT OUT1,AL ;OUTPUT DEL BIT DI CONTROLLO
0004 BB5A04    12  MOV BX,#045AH ;CARICA BX COL VALORE COUNT
0007 4B        13  BACK    DEC BX          ;BX = BX - 1
0008 75FD      14  JNZ BACK                ;BX ≠ 0, DEC NUOVAMENTE
000A FEC8      15  DEC AL                  ;AL = 0
000C E658      16  OUT OUT1,AL             ;BIT DI CONTROLLO=0
               17 ;
               18 ;   ;IMPULSO RITORNATO A UNO 0 LOGICO
```

Figura 7.17 — Un programma 8086/8088 per generare un impulso usando la tecnica di ritardo software mostrata in Figura 7.16.

Ritardi più lunghi

Con l'8086/8088 possiamo facilmente generare ritardi usando registri o a 8 o a 16 bit. Tuttavia, se abbiamo bisogno di un ritardo più lungo, possiamo usare due registri da 16 bit ed "innestare" i cicli di decremento (come mostrato nel diagramma di flusso in Figura 7.18). Un programma per realizzare questo diagramma di flusso è mostrato nella Figura 7.19.

USO DI UN DISPOSITIVO PIO 8255 CON IL SISTEMA 8086/8088

Illustreremo ora come usare l'8086/8088 per controllare l'8255, un dispositivo programmabile di input/output. Tali dispositivi vengono spesso usati in un sistema microprocessore per permettere alla CPU di controllare elettricamente l'hardware periferico. Inizieremo con una breve spiegazione dell'8255; quindi procederemo all'esame di alcuni esempi campione di routine software, scritte nel linguaggio assembly dell'8086/8088, per il controllo del dispositivo.

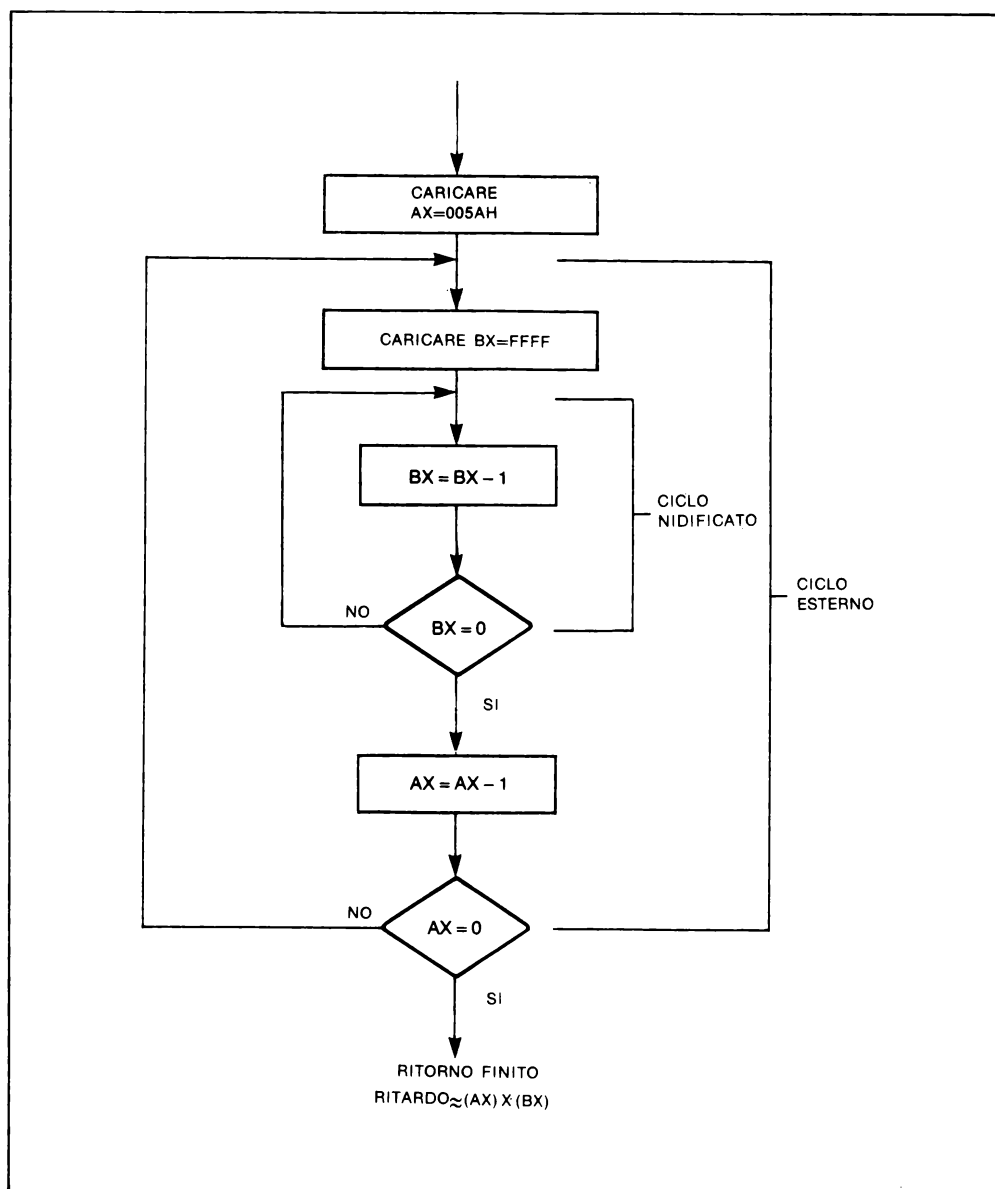



Figura 7.18 — Il diagramma mostra come si può generare un ritardo più lungo usando due registri interni della CPU.

Illustrazione dell'8255

L'8255 è un dispositivo LSI a 40 pin DIP (Dual In-line Package), progettato per eseguire una varietà di funzioni di inter-

```
                 1  ;
                 2  ;  Programma per generare un ritardo
                 3  ;  Usando 2 registri, AX e BX.
                 4  ;  AX è il loop esterno, BX è il loop interno.
                 5  ;
0000 B85A00      6  MOV AX,#0005AH ;Carica il registro AX.
0003 BBFFFF      7  BACK      MOV BX,#0FFFFH ;Carica il registro BX
0006 4B          8  BACK1     DEC BX             ;BX = BX - 1
0007 75FD        9  JNZ BACK1          ;loop interno≠ 0
0009 48         10  DEC AX             ;AX = AX - 1
000A 75F7       11  JNZ BACK           ;loop esterno≠ 0
                12  ;
                13  ;  ;loop di ritardo approssimativamente uguale ad AXxBX.
```

Figura 7.19 — Programma 8086/8088 per realizzare un loop di ritardo come mostrato in Figura 7.18.

faccia nell'ambiente di un sistema microprocessore.

La Figura 7.20 mostra un diagramma a blocchi del dispositivo 8255. Esaminiamo questo diagramma e discutiamo in generale la funzione di ogni blocco.

Nel diagramma troviamo 4 blocchi che collegano fisicamente l'8255 all'hardware esterno. Tali blocchi hanno le linee di controllo etichettate PA0-PA7, PB0-PB7, PC0-PC3, PC4-PC7. I gruppi di segnali provenienti da questi blocchi sono divisi logicamente in 3 porte di I/O differenti etichettate PORTA (PA), PORTB (PB) E PORTC (PC).

Porte di I/O dell'8255

Questi 4 blocchi di porte (PC si divide in due gruppi) sono connessi ad un canale di dati interno al dispositivo 8255.

È attraverso questo bus dati interno che le porte vengono programmate.

In Figura 7.20 troviamo 2 blocchi. Questi blocchi, etichettati controllo gruppo A e controllo gruppo B, definiscono come le tre porte di I/O devono operare nel sistema. Esistono diversi modi operativi per l'8255, ed essi devono essere definiti attraverso la scrittura da parte della CPU di parole di controllo nel dispositivo. Si noti che il gruppo C dell'8255 è costituito da due porte a 4 bit. Uno dei gruppi a 4 bit è associato con i segnali del dispositivo di gruppo A e l'altro con i segnali del dispositivo di gruppo B.

Gli ultimi blocchi logici in Figura 7.20 sono "buffer del bus dati" e "logica di controllo lettura/scrittura". Questi due blocchi forniscono l'interfaccia elettrica tra i microprocessori 8086/8088 e l'8255.

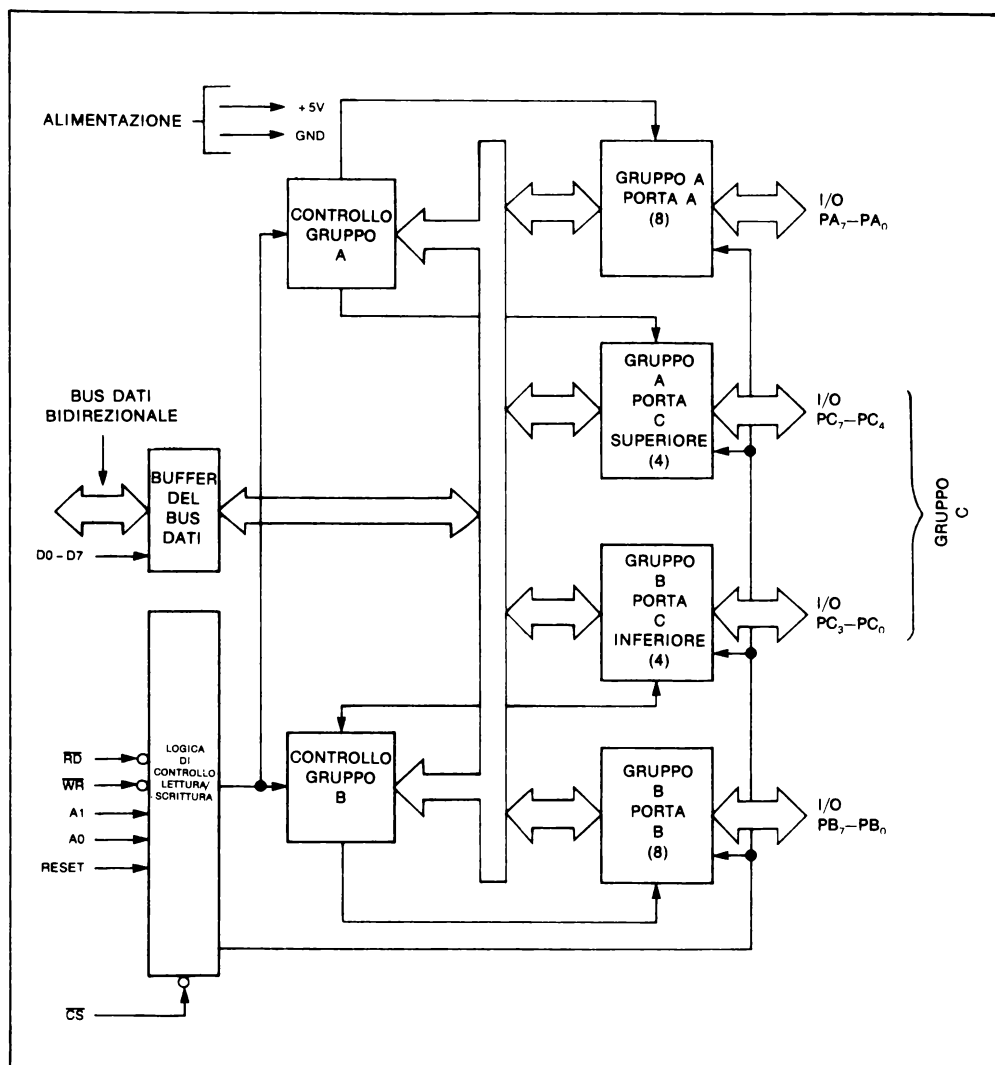



Figura 7.20 — Il diagramma mostra il PIO 8255.

Funzioni del buffer e della logica di controllo

Il buffer del bus dati realizza un buffer tra le linee dati di input e di output ed il bus dati della CPU.

La logica di controllo lettura/scrittura esegue il corretto smistamento dei dati diretti ai registri interni o provenienti da essi, garantendo inoltre l'esatta sincronizzazione. Tale sincronizzazione dipende dal tipo di operazione eseguita dalla CPU; cioè dipende dal fatto che sia o meno un'operazione I/O di lettura o di scrittura.

Collegamento fisico al sistema CPU

La Figura 7.21 mostra un diagramma a blocchi che illustra in che modo il sistema 8255 è connesso al sistema 8086/8088. Si noti che ci sono 4 porte interne a cui accedono le linee di indirizzo A1 e A0. Useremo gli indirizzi di porta 10H, 11H, 12H, e 13H per l'8088, e 10H, 12H, 14H, e 16H per il sistema 8086. La differenza fra questi due insiemi di indirizzi consiste nel fatto che ogni volta che si scrive dall'8086 in un indirizzo dispari, il sistema trasferisce i dati sulle linee dati superiori (D8-D15). Percio' quando ci si collega con le linee dati inferiori (D0-D7) dell'8086, tutti gli indirizzi di porta devono essere pari.

I registri di lettura e scrittura dell'8255

Esaminiamo ora le definizioni del registro e gli indirizzi delle porte per il sistema 8255 illustrato in Figura 7.21. Supporremo che la CPU sia una 8088.

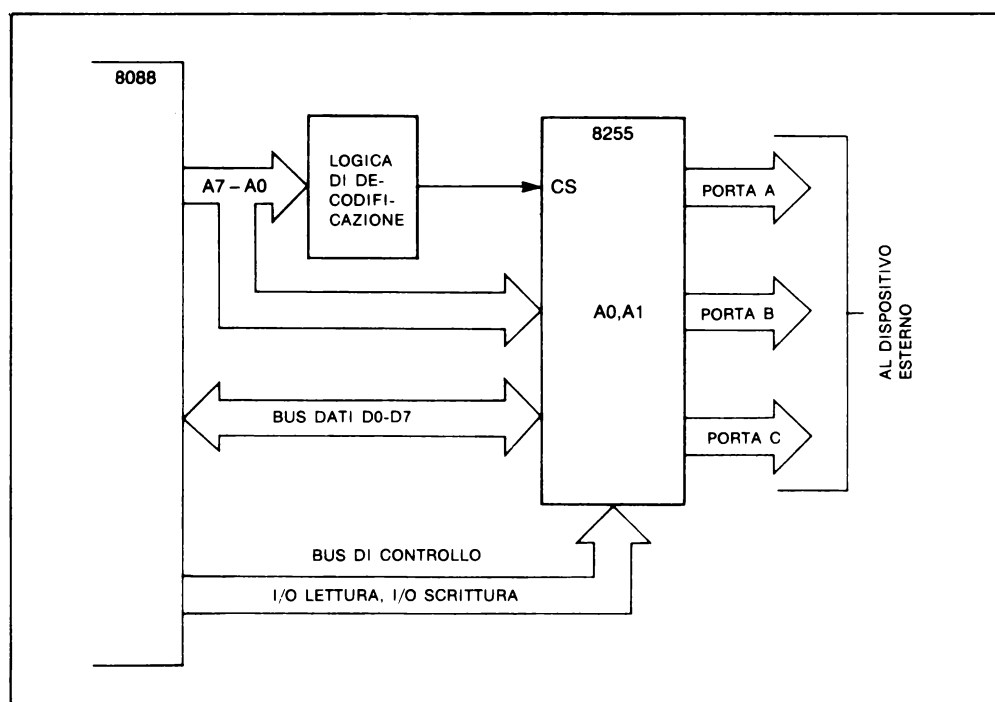



Figura 7.21 — Il diagramma mostra come l'8255 si colleghi con la CPU 8088. Se l'8255 deve collegarsi con l'8086, si deve decidere quale byte del bus dati a 16 bit debba essere collegato.

Elenchiamo qui le definizioni di registri:

PORTA #	DEFINIZIONE	DESCRIZIONE
10H	scrive i dati alla porta A	uscita
10H	legge i dati alla porta A	ingresso
11H	srive i dati alla porta B	uscita
11H	legge i dati alla porta B	ingresso
12H	scrive i dati alla porta C	uscita
12H	legge i dati alla porta C	ingresso
13H	scrive parola di controllo	uscita
13H	registro di lettura non consentita	

La funzione dei registri 0-2 è definita dalla parola scritta nel registro di controllo 3 del 8255. La Figura 7.22 mostra le definizioni dei bit del registro di controllo.

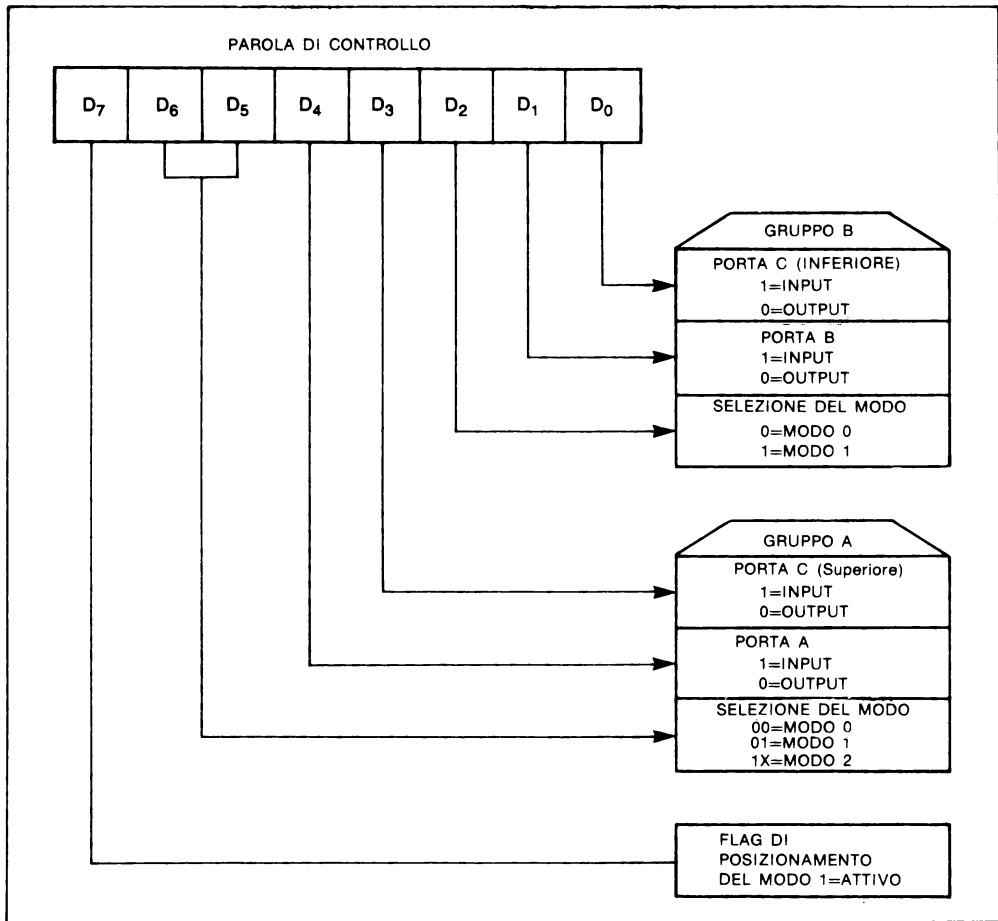



Figura 7.22 — Definizioni dei bit per programmare la parola di controllo dell'8255.

Dovremo far riferimento a queste definizioni quando scriveremo un programma per usare l'8255.

Input e output di base con il sistema 8255

Per usare l'8255 nel modo base di I/O (input/output), il programmatore dovrà innanzitutto scrivere un byte nel registro di controllo. Questo byte definisce come andranno adoperati i registri del dispositivo 8255. I bit del registro di controllo usati per programmare la funzione base di I/O saranno disposti come segue:

1	0	0	0	0	0	0	0
D7	D6	D5	D4	D3	D2	D1	D0

Significato dei bit della parola di controllo

Guardando la Figura 7.22 vediamo che il bit D7 definisce questo byte come una parola di controllo per configurare il modo di operazione. Esaminiamo ogni bit:

I bit D6 e D5 definiscono le modalità di operazione per la porta A dell'8255. Questo sarà il modo 0.

Il bit D4 = 0 indica che la porta A è un'uscita (output)

Il bit D3 = 0 posiziona i 4 bit superiori della porta C come uscita.

Il bit D2 = seleziona la modalita' operativa della porta B.
Uno zero logico seleziona il modo 0.

Il bit D1 = 0 posiziona la porta B come porta di uscita.

Il bit D0 = 0 posiziona i 4 bit inferiori della porta C come porta d'uscita.

Con questa parola di controllo scritta nell'8255, tutte e 3 le porte (A, B e C), vengono definite come porte di output.

Ciò permette all'8255 di avere 24 linee di output separate da usare per interfacciare con i dispositivi esterni. Le istruzioni dell'8086/8088 per posizionare l'8255 in questo modo sono le seguenti:

```
MOV AL,10000000B    PAROLA DI CONTROLLO IN AL
OUT 13H,AL          OUTPUT DELLA PAROLA DI
                    CONTROLLO ALL'8255
```

Una volta che l'8255 è stato programmato con queste due istruzioni, possiamo semplicemente scrivere i dati di output dell'8086/8088. Supponiamo, ad esempio, di voler scrivere 23H sull'uscita della porta A, 41H sull'uscita della porta B, e 73H sull'uscita della porta C. Per soddisfare tale esigenza, avremo bisogno di una sequenza di istruzioni come la seguente:

```
MOV AL,23H          POSIZIONA I DATI
                    PER LA PORTA A
```

OUT 10H,AL	OUTPUT DEI DATI ALLA PORTA A
MOV AL,41,H	POSIZIONA I DATI PER LA PORTA B
OUT 11H,AL	OUTPUT DEI DATI ALLA PORTA B
MOV AL,73H	POSIZIONA I DATI PER LA PORTA C
OUT 12H,AL	OUTPUT DEI DATI ALLA PORTA C

Quando avremo eseguito tali istruzioni, le porte di uscita A, B e C dell'8255 risulteranno programmate secondo i valori dei dati specificati. Questo è un modo conveniente per ottenere tre singole porte di uscita contenute in un unico chip.

Si potrebbero programmare le porte per una combinazione

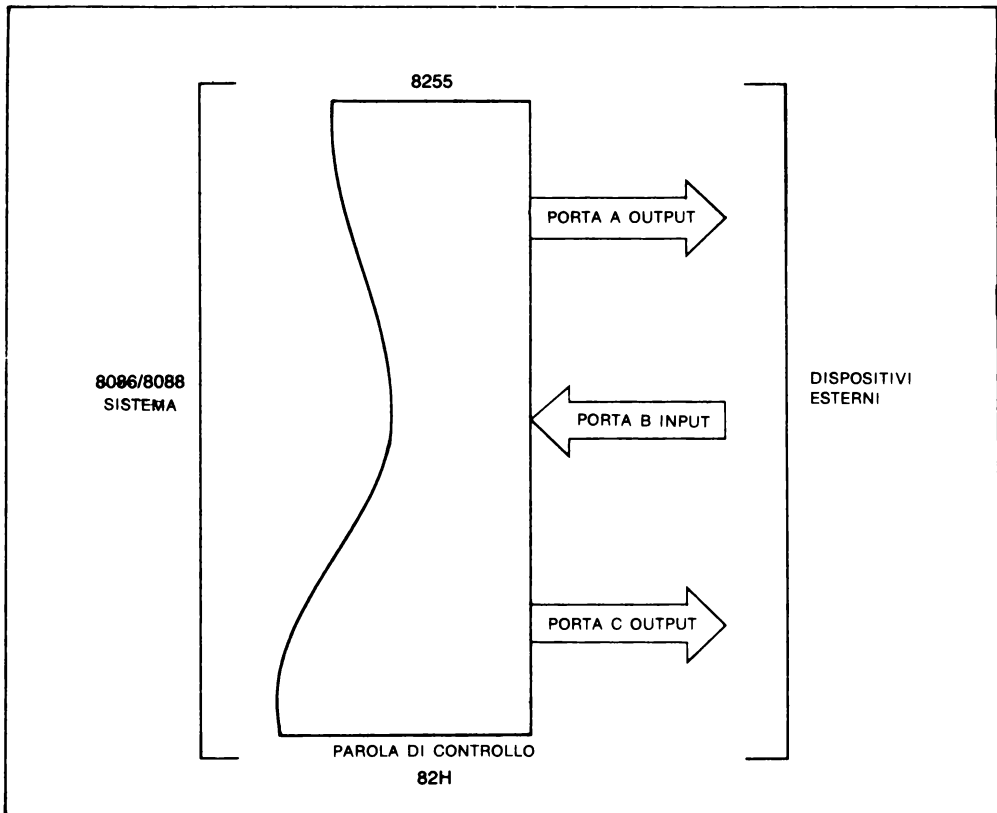



Figura 7.23 — Il diagramma mostra la configurazione dell'8255 dopo che vi è stata scritta la parola di controllo.

di input e di output. Si potrebbero, ad esempio, programmare le porte A e C in modo che siano porte di uscita, e la porta B in modo che sia una porta di entrata.

In base alla Figura 7.22, la parola di controllo per posizionare l'8255 in questa configurazione sarà allora:

D7							D0
1	0	0	0	0	0	1	0

Dopo aver programmato la parola di controllo nel registro 3 dell'8255, il dispositivo risulterà posizionato nello stato logico prestabilito (come mostato nella Figura 7.23). Un'istruzione IN potrà venir usata per leggere i dati di input dalla porta B:

```
IN AL,11H          LEGGE I DATI DALLA
                   PORTA B
```

Una volta che l'80255è posizionato nella configurazione corretta, si potranno facilmente leggere o scrivere dati su qualsiasi porta. Esistono molte combinazioni differenti per posizionare l'8255 nel modo 0 della configurazione base di I/O. Fin qui ne abbiamo nominate solo alcune.

ESEMPIO DI ANALISI DI UNA TASTIERA

Come esempio d'uso dell'8086/8088 con l'8255, analizziamo ora un'interfaccia per una tastiera. Supponiamo che la tastiera sia organizzata secondo una matrice 4 × 4 di interruttori unipolari (SPST), e usiamo l'8255 per interfacciare gli interruttori della tastiera con l'8088. La Figura 7.24 mostra un diagramma a blocchi di questo sistema.

Algoritmo di rilevazione della pressione di un tasto

Le uscite della porta B sono connesse alle colonne della tastiera. Gli ingressi della porta A sono connessi alle righe. Le colonne vengono forzate ad uno zero logico una alla volta. Dopo che ciascuna colonna è stata posizionata sullo zero logico, la porta di ingresso A viene letta. Se un bit qualsiasi letto dalla porta di ingresso è uno zero logico, vorrà dire che è stato premuto un tasto. Basandoci sulla colonna, che è uno zero logico, e sul bit nella porta di ingresso, che è uno zero logico, saremo in grado di sapere quale tasto della matrice è stato premuto. La Figura 7.25 mostra un programma che opera nella maniera appena descritta.

Benchè questo esempio sia stato formulato per una matrice piccola, (4 × 4), lo si può facilmente espandere per una matrice di qualsiasi dimensione. Usando l'8255 si fornisce, in modo facile, l'hardware necessario per l'interfaccia.

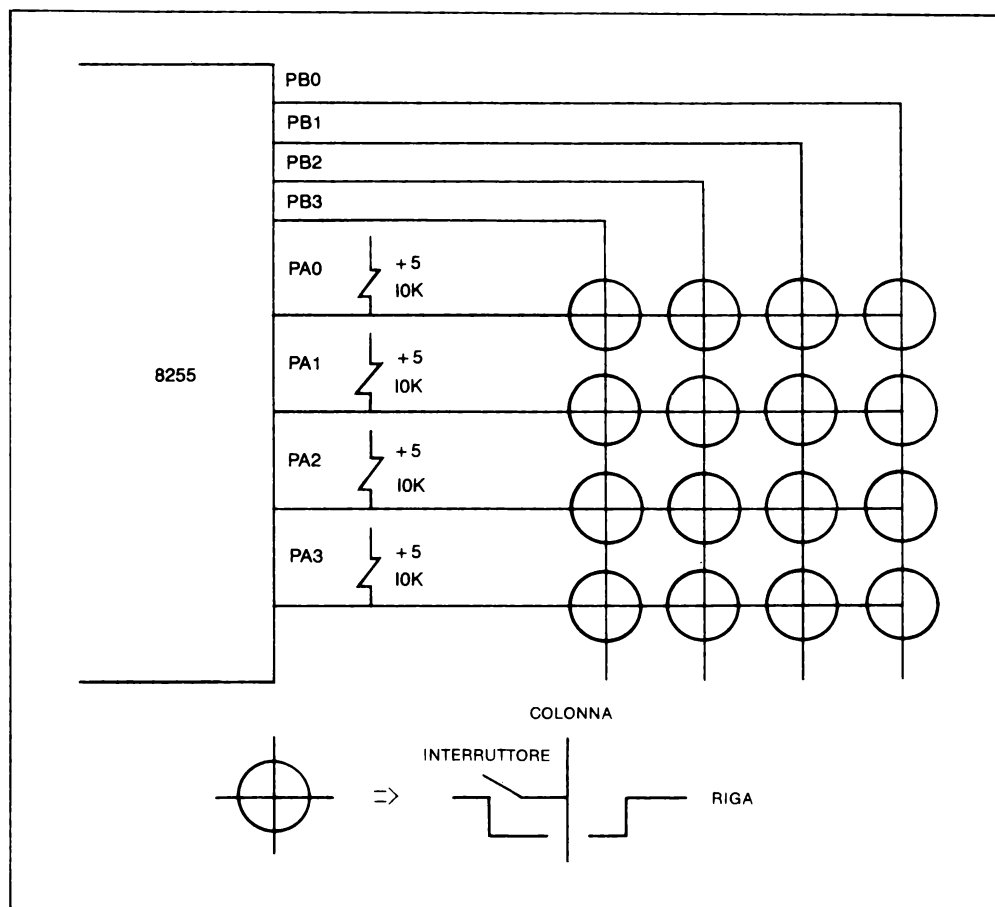



Figura 7.24 — Un diagramma schematico che mostra le interconnessioni per una semplice tastiera 4×4 interfacciata ad un 8255.

Programmazione del chip temporizzatore 8253

In questa sezione mostreremo come l'8086/8088 possa venir usato per controllare un chip temporizzatore programmabile.

Il chip temporizzatore programmabile viene usato in molte applicazioni di sistema dove sono necessarie funzioni di temporizzazione. Si ricordi come in un'altra sezione di questo capitolo sia stato illustrato il modo per generare un'impulso usando il software; avevamo raggiunto questo scopo usando il microprocessore in un ciclo (loop) software.

Vantaggio dell'uso dell'8253 per la generazione di un impulso

Qualora si usi un chip temporizzatore, il microprocessore avrà bisogno solamente di scrivere pochi byte di controllo per

posizionare i parametri di tempo, e il chip farà il resto del lavoro. Questo permette al microprocessore di eseguire altre funzioni nel sistema.

```
              1  ;
              2  ;  PROGRAMMA PER RILEVARE LA PRESSIONE DI UN TASTO
              3  ;  PER LA MATRICE MOSTRATA IN FIGURA 7-24
              4  ;
              5  ;  LA PORTA B È LA PORTA DI OUTPUT
              6  ;  LE PORTE A E C SONO LE PORTE DI INPUT
              7  ;
              8  PORTA    EQU 10H
              9  PORTB    EQU 11H
             10  PORTC    EQU 12H
             11  CONP     EQU 13H      ;PORTA DI CONTROLLO
             12  CONWD    EQU 99H      ;PAROLA DI CONTROLLO
             13  ;
             14  ;
0000 B099    15           MOV AL,#CONWD     ;CARICA CONWD IN AL
0002 E613    16           OUT CONP,AL       ;CONWD ALLA PORTA DI CONTROLLO DELL'8255
             17  ;
0004 BA1100  18           MOV DX,#PORT B    ;DX=INDIRIZZO DI PORTA B
             19  ;
0007 B3FE    20  COL      MOV BL,#0FEH      ;COLONNA 0 ATTIVA
0009 88D8    21  COL1     MOV AL,BL         ;COLONNA ATTIVA AD AL PER L'OUTPUT
000B EE      22           OUT DX,AL         ;INVIA COLONNA ATTIVA
000C E410    23           IN AL,PORTA       ;LEGGE LE LINEE DELLE RIGHE
000E F6D0    24           NOT AL            ;COMPLEMENTA LA PAROLA DI INPUT
0010 240F    25           AND AL,#0FH       ;MASCHERA I BIT NON USATI
0012 3C00    26           CMP AL,#00H       ;QUALCHE 1 ATTIVO?
0014 750A    27           JNZ KEYIN         ;SE SÌ, ALLORA TASTO PREMUTO
0016 D0C3    28           ROL BL,1          ;RUOTA A SINISTRA DI 1 BIT LA COLONNA ATTIVA
0018 80FBEF  29           CMP BL,#0EFH      ;ASSERITA ULTIMA COLONNA?
001B 75EC    30           JNZ COL1          ;NON L'ULTIMA COLONNA, ASSERIRE LA SUCCESSIVA
001D E9E7FF  31           JMP COL           ;ULTIMA COLONNA, ASSERISCE PRIMA COLONNA
             32  ;
             33  ;    IL SEGUITO TRATTA LA CHIUSURA DI UN TASTO
             34  ;
0020 88C1    35  KEYIN    MOV CL,AL         ;POS RIGA=1 NEL REGISTRO CL
0022 88D8    36           MOV AL,BL         ;POS COLONNA NEL REGISTRO A
0024 F6D0    37           NOT AL            ;POS COLONNA=1 NEL REGISTRO A
0026 240F    38           AND AL,#0FH       ;BIT NON USATI=0
0028 C3      39           RET               ;RITORNO DALLA SUBROUTINE DI SCANSIONE
             40
             41     ;RITORNA DALLA ROUTINE CON
             42     ;REG AL=POS COLONNA, 1 NEL BIT COLONNA ATTIVA
             43     ;REG CL=POS RIGA, 1 NEL BIT COLONNA ATTIVA
             44
```

Figura 7.25 — Questo programma per l'8086/8088 controlla la tastiera a matrice 4×4 mostrata in Figura 7.24.

Diagramma a blocchi del chip temporizzatore 8253

La Figura 7.26 mostra un diagramma a blocchi dell'8253. Dalla figura si può vedere che ci sono 3 contatori programmabili indipendenti. Dato che i contatori sono identici non li analizzeremo individualmente. Concentreremo invece la nostra attenzione sulla programmazione di un singolo contatore.

Questa informazione può essere quindi usata per programmare uno qualsiasi dei tre contatori. Un blocco importante in

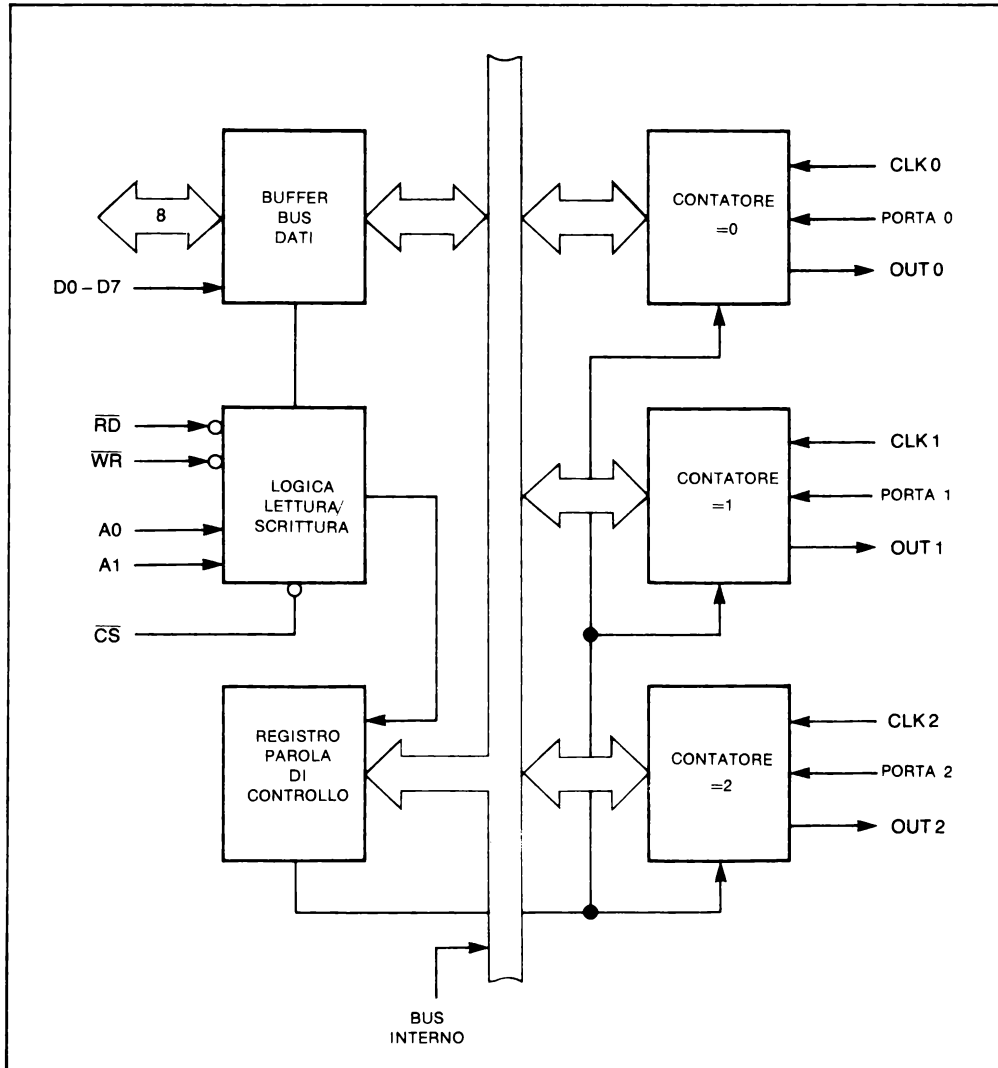



Figura 7.26 — Questo diagramma mostra il chip temporizzatore programmabile 8253.

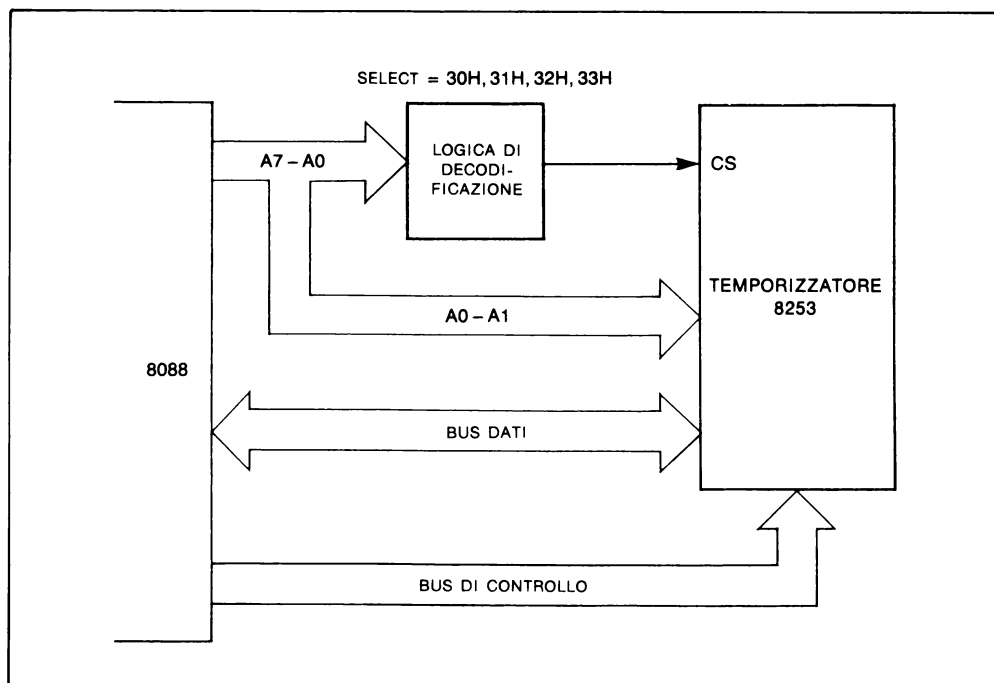



Figura 7.27 — Questo diagramma mostra come collegare l'8253 con la CPU 8088.

Figura 7.26 è costituito dal buffer del bus dati. Questa è la logica che collega il bus dati del sistema da e per il microprocessore ai registri interni dell'8253. Il blocco che controlla la lettura e la scrittura dei registri contatori è chiamato logica di lettura/scrittura.

Funzione del registro di controllo dell'8253

Il blocco finale in Figura 7.26 è chiamato registro della parola di controllo. Questo registro contiene l'informazione programmata mandata al dispositivo dal microprocessore del sistema. In effetti, questo registro definisce come il dispositivo deve operare logicamente. La Figura 7.27 presenta un diagramma a blocchi che mostra come l'8253 si connetta con il sistema 8088.

Generazione di un impulso programmabile

One-shot: generazione di un impulso di durata fissata

Riferendoci al diagramma a blocchi della Figura 7.27 si può vedere che gli indirizzi delle porte per i dispositivi sono 30H, 31H, 32H, 33H, dove 33H è la porta di controllo e 30H, 31H, 32H sono rispettivamente i registri contatori 0, 1 e 2.

Programmiamo ora l'8253 per generare un impulso di una

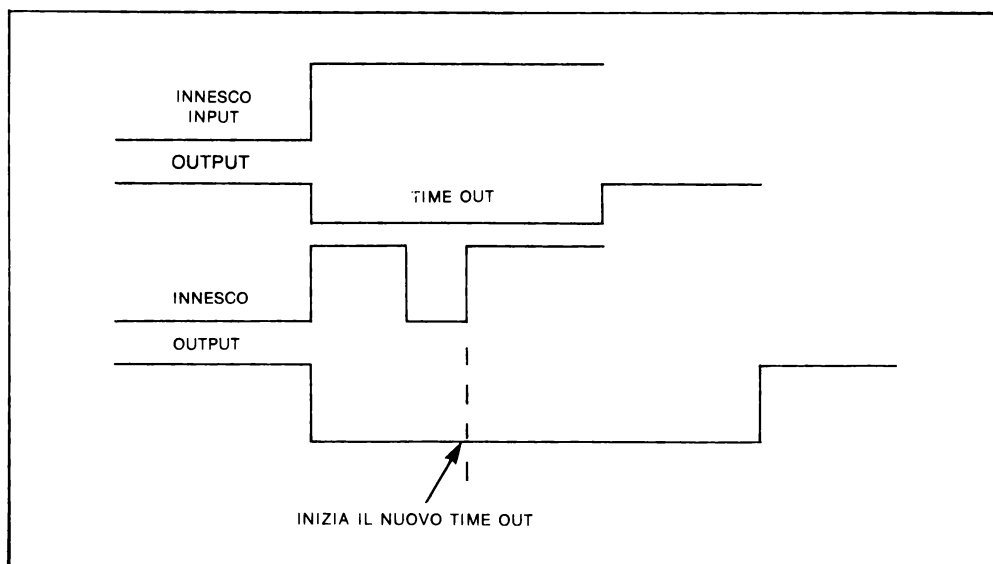



Figura 7.28 — Questo diagramma di temporizzazione mostra il modo in cui l'8253 può essere usato come "one-shot" programmabile. Quando l'input di innesco va ad 1 logico, l'output va ad uno 0 logico per il tempo programmato dal contatore. Se l'innesco viene applicato nuovamente prima che sia trascorso il tempo programmato, il conto verrà reinnescato dall'inizio.

durata fissata sotto il controllo del programma. Questa azione talvolta viene chiamata "one-shot".

Nel modo di impulso programmabile, il dispositivo 8253 può essere configurato in modo da dare un impulso di uscita che sia un numero intero di impulsi di clock. L'one-shot verrà innescato sul fronte di salita dalla porta di input; se gli inneschi avvengono durante l'output dell'impulso il dispositivo verrà reinnescato, come mostrato in Figura 7.28.

Supponiamo che la frequenza del clock di ingresso all' 8253 sia un megahertz. Programmeremo l'8253 per un impulso di uscita di esattamente 75 microsecondi. Il programma dell' 8086/8088 per eseguire questa funzione è mostrato in Figura 7.29. In questa figura si suppone che il contatore N. 1 sia usato come one-shot programmabile.

SOMMARIO

In questo capitolo abbiamo presentato i punti essenziali delle operazioni di input e di output dell'8086/8088.

Abbiamo cominciato definendo i termini input e output.

```
                 1 ;
                 2 ;  ;PROGRAMMA PER USARE L'8253 COME
                 3 ;  ;ONE-SHOT PROGRAMMABILE
                 4 ;
                 5 CNTR1        EQU 31H        ; INDIRIZZO PORTA CONTATORE 1
                 6 CONP         EQU 33H        ;INDIRIZZO PORTA DI CONTROLLO
                 7 ;
0000 B072        8              MOV AL,#72H         ;;PAROLA DI CONTROLLO AD AL
0002 E633        9              OUT CONP,AL         ;;PAROLA DI CONTROLLO ALL'8253
                10 ;
                11 ;  CTR 1, RL = 3, M = 1, CONTATORE BINARIO
                12 ;
0004 B04B       13              MOV AL,#75          ;75 DECIMALE
0006 E631       14              OUT CNTR1,AL ;NUMERO AL CONTATORE 1, BYTE MENO SIGNIF.
0008 B000       15              MOV AL,#00
000A E631       16              OUT CNTR1,AL   ;OUTPUT AL CONTATORE 1, BYTE PIÙ SIGNIF.
                17 ;
                18 ;
                19 ;  IL DISPOSITIVO È ORA PREDISPOSTO AD ASPETTARE
                20 ;  L'INPUT DI INNESCO AL CONTATORE
                21 ;
```

Figura 7.29 — Programma 8086/8088 per usare l'8253 come one-shot programmabile con una ampiezza d'impulso di 75 microsecondi.

Quindi abbiamo analizzato le due istruzioni di I/O dell'8086/8088: IN e OUT. Abbiamo visto che entrambe queste istruzioni possono essere usate con porta fissa o con porta variabile. Sono stati dati degli esempi. Infine abbiamo esaminato due esempi di programmazione di chips periferici speciali LSI, chips che sono abbastanza frequenti in molti sistemi 8086/8088. Descrivendo ogni chip abbiamo esaminato la decodifica di una porta e un diagramma a blocchi delle sue connessioni con il sistema.

Nel capitolo 8 continueremo la nostra discussione delle applicazioni dell'8086/8088 e scriveremo dei programmi per l'8088 che possono essere usati con il BIOS (sistema base di input/output) del Personal Computer IBM. La comprensione dei concetti presentati in questo capitolo vi aiuterà a scrivere programmi per il prossimo capitolo.

CAPITOLO 8

LA FAMIGLIA DEI PC IBM

Oggi l'IBM è il più grande produttore di PC nel mondo. Dal PC IBM, introdotto nel 1981, la famiglia dei PC è cresciuta in potenza e varietà.

Tra i PC prodotti, ora vi mostriamo alcuni esempi, anche se non tutti sono più in produzione:

Modelli di PC

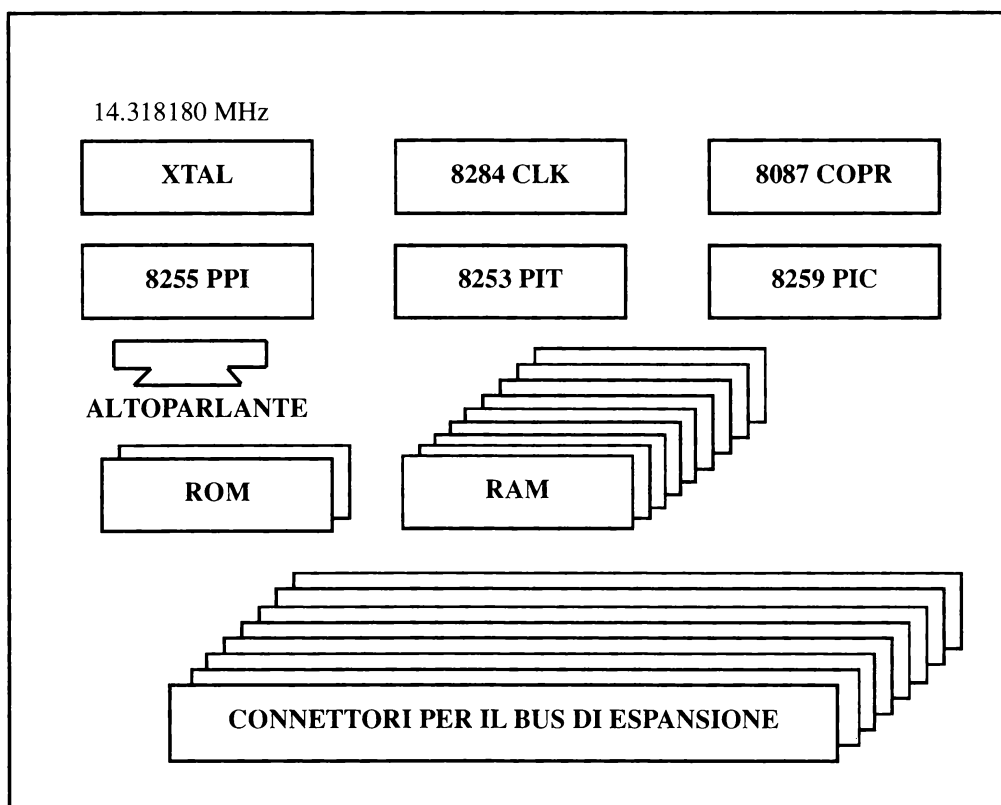
Il PC	è basato sul microprocessore 8088.
Il PC XT	aggiunge alle facilità del PC un disco fisso, più slot di espansione, e un alimentatore più potente.
Il PCjr	è una versione limitata del PC prodotto per il mercato degli home computer.
Il Portable PC	usa la piastra di sistema del PC XT ma non ha tutte le sue periferiche.
Il PC XT/370	è una combinazione di un PC XT, un terminale IBM 3270 e una workstation per un mainframe IBM 370.
Il 3270 PC	è un altro prodotto combinazione, questa volta per mainframe IBM 43XX e 308X.
Il PC AT	è basato sul microprocessore 80286 con la possibilità di gestire memoria sopra il limite dell'8088.
Il PC Convertible	un PC portatile con un schermo LCD.
La famiglia PS/2	usano i microprocessori 80286 e 80386, dischi di 3,5 pollici e, con l'eccezione dei modelli base, vengono forniti con il nuovo bus MCA (Micro Channel Architecture).

A questa piccola lista dobbiamo aggiungere i tantissimi PC compatibili fabbricati nei diversi paesi del mondo.

PC compatibili

Il fatto che tutte queste macchine sono compatibili (più o meno) con il PC IBM originale ci offre la possibilità di scrivere un programma valido per tutti i PC. Bisogna solo conoscere le regole di programmazione particolari del PC per mantenere la portabilità del programma. Quindi, è possibile vendere il software in un mercato più vasto.

ARCHITTETURA HARDWARE DEL PC

La Piastra del Sistema

Struttura interna

La struttura di tutti i computer della famiglia PC IBM è stata ideata in modo da comprendere tutti i componenti principali sulla piastra del sistema. Nel caso del PC, questa piastra contiene il microprocessore 8088, con almeno 64 Kbyte di memoria, il BIOS ed il BASIC nella ROM ed i chip ausiliari che controllano i dischi, lo schermo, la tastiera, ecc. In

più, vi sono i connettori del bus di espansione e uno zoccolo per il coprocessore matematico 8087.

I Bus

Ogni chip sulla piastra del sistema è collegato direttamente o indirettamente con i bus degli indirizzi, dei dati e di controllo e con le linee di alimentazione.

Struttura del bus degli indirizzi

Il bus degli indirizzi ha 20 linee di segnale. Ogni linea può assumere i valori 0 o 1, quindi il bus può indirizzare 1 Mbyte (1.048.576) di locazione nella memoria. Queste linee sono usate anche per le porte I/O di entrata e di uscita. L'8088 utilizza soltanto le 16 linee più basse per indirizzare le porte. Quindi sono un massimo di 64 Kbyte (65536) porte.

Bus dati

L'8088 ha un bus dati esterno di 8 linee. Tuttavia i bus interni sono a 16 bit, il trasferimento dei dati all'esterno avviene in due fasi da 8 bit ciascuna.

Bus di controllo

La sincronizzazione dei chip avviene tramite linee di controllo e di status. Lasciamo questa parte al progettista dell'hardware.

Il microprocessore 8088

Abbiamo già trattato l'argomento della CPU nella prima parte del libro.

Il coprocessore matematico 8087

Velocità

Per gli applicativi numerici risulterebbe troppo lento utilizzare delle subroutine scritte per l'8088, il quale lavora soltanto con numeri interi.
Per ottenere una velocità più elevata bisogna inserire un chip 8087 nello zoccolo sulla piastra del sistema.

Coprocessore

L'8087 ha la capacità di effettuare operazioni aritmetiche, trigonometriche, esponenziali e logaritmiche, con interi di 16, 32 e 64 bit, numeri in virgola mobile di 32, 64 e 80 bit e numeri decimali con 18 cifre. L'8087 aggiunge quasi 70 nuove istruzioni a quelle dell'8088 e otto registri di 80 bit. In più, può eseguire queste operazioni autonomamente mentre l'8088 svolge un altro lavoro.

Il gestore degli interrupt 8259 (il PIC)

Quando una periferica vuole scambiare dati con la CPU deve avvisarla per richiamarne l'attenzione. Questo è fatto tramite un interrupt.

Interrupt

L'8259 intercetta gli interrupt, determina il loro livello di importanza in relazione agli altri interrupt ed invia un interrupt alla CPU. Poi l'8088 sospende l'esecuzione del programma principale e, tramite i dati provenienti dall'8259, esegue una routine di servizio dell'interrupt. Una volta terminato, il microprocessore ritornerà al programma principale.

Generalmente è meglio non programmare l'8259, per evitare problemi con l'attività di base del PC.

Il gestore del DMA 8237

Direct Memory Access

DMA (Direct Memory Access) permette ad una periferica, ad esempio il lettore di dischi, di trasferire dati tra memoria e periferica senza passare attraverso la CPU. In questo modo l'operazione dei dischi è più veloce.

Il generatore di temporizzazione 8284

Temporizzazione

L'8284 fornisce un segnale di reset del sistema e i segnali per sincronizzare i chip del PC. Una frequenza di riferimento è divisa per una costante in modo da ottenere una frequenza di 4.77 MHz nel PC originale. I nuovi PC usano delle frequenze più alte.

L'interfaccia periferica programmabile 8255 (il PPI)

I/O port

Questo chip viene programmato dal BIOS per controllare alcune periferiche. Normalmente è programmato dal software applicativo solo per utilizzare l'altoparlante del PC.

Il temporizzatore programmabile 8253 (il PIT)

Contatori del timer

Il temporizzatore programmabile ha come entrata una frequenza di riferimento di 1.193182 MHz. Il temporizzatore contiene 3 contatori da

16 bit:

Il contatore 0 genera un interrupt ad intervallo costante per implementare un orologio. Il BIOS setta un divisore di 65536, per avere un risultato di 18.2 interrupt per secondo.

Il contatore 1 è riservato ai cicli di refresh della memoria.

Il contatore 2 è disponibile per uso generale, soprattutto come generatore di toni per l'altoparlante del PC.

Il gestore dell'interfaccia seriale 8250 UART

UART per RS-232

Questa periferica viene programmata dal BIOS per comunicare tramite la porta seriale RS-232 in polling mode, e questo limita la velocità di trasmissione perche è necessario che il BIOS richieda di frequente la disponibilità dei dati alla porta. E' possibile scrivere un programma che usa l'8250 in modo interrupt per comunicare con una velocità di trasmissione molto superiore.

Indirizzi input/output delle periferiche

Indirizzi di I/O

0000	Il gestore del DMA 8237.
0020	Il gestore degli interrupt 8259.
0040	Il temporizzatore 8253.
0060	L'interfaccia parallela 8255.
0080	Registro pagina DMA.
00A0	Registro maschera NMI.
0200	Il Controller giochi.
0210	Unità di espansione.
02F8	Porta seriale 2.
0320	Disco fisso.
0378	Porta parallela 1.
03B0	Video monocromatico.
03D0	Video colore.
03F0	Dischetti floppy.
03F8	Porta seriale 1.

Accesso diretto all'hardware

Uso dell'hardware

Normalmente è pericoloso scrivere un programma che accede direttamente all'hardware. Può causare dei problemi con altri programmi o non funzionare sui PC compatibili. Dove possibile è meglio usare la funzione del BIOS, o meglio ancora, del DOS.

In alcuni casi è necessario usare l'hardware. Ad esempio nei programmi dove la velocità è importante. Un altro esempio è quando il BIOS non prevede servizi per una parte dell'hardware, quindi non ci sono alternative.

Queste periferiche sono tra le più usate:

Periferiche hardware

I contatori del PIT.
Il controller degli interrupt PIC.
L'8250 UART.
Il controller video 6845.
Controllo dell'altoparlante.
Il controller dei giochi.

L'Altoparlante

Il controllo del sistema suono non è possibile tramite il BIOS o il DOS. E' necessario accedere direttamente all'hardware, in questo caso al 8255 PPI e al contatore 2 del 8253 PIT.

PPI bit 0 abilita la frequenza di riferimento al contatore 2.
PPI bit 1 abilita l'uscita del contatore 2.

Controllo altoparlante

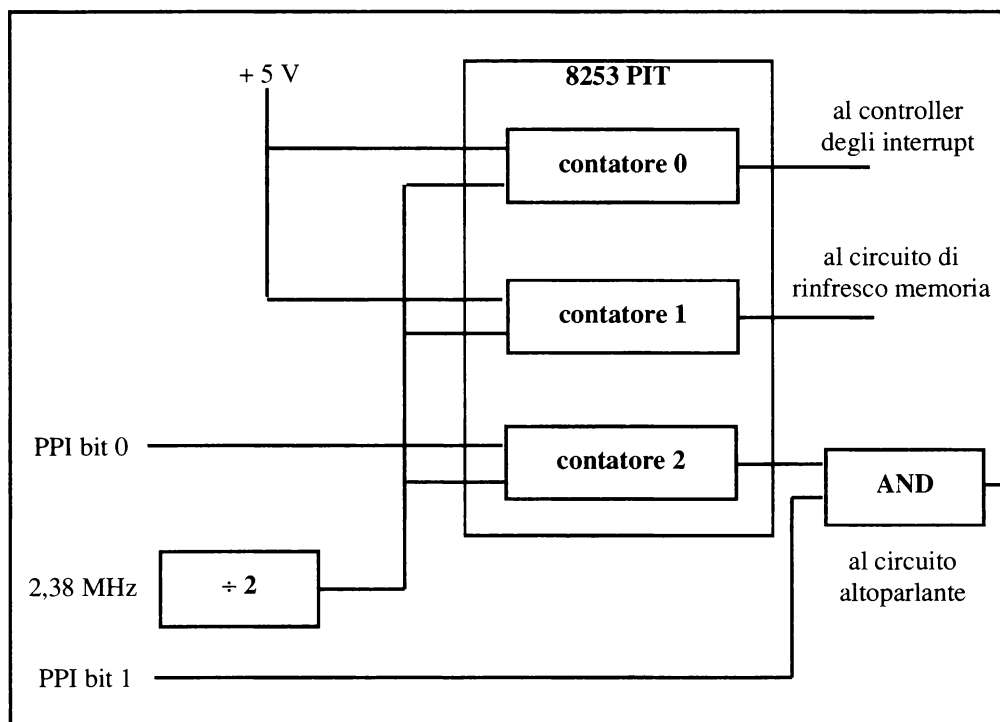
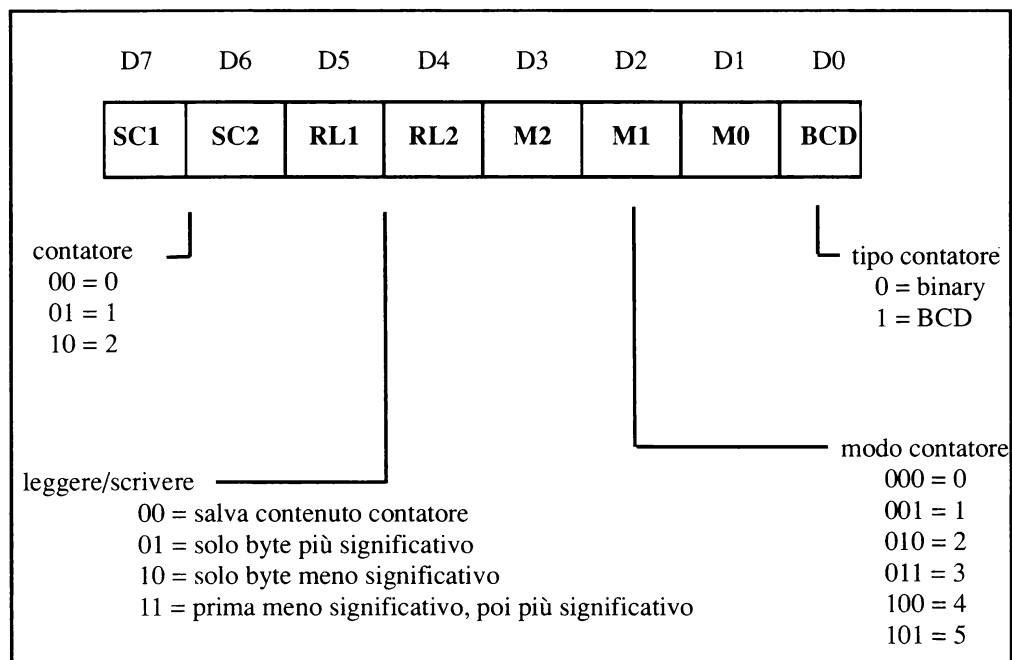
Quando i bit 0 e 1 del PPI sono settati al livello 1, tramite un OR di 03h con i bit della porta 61h, l'uscita del contatore 2 del PIT è mandata all'altoparlante. Una AND di FCh con i bit della porta 61h interrompe il suono.

Così possiamo far suonare l'altoparlante, ma per cambiare la frequenza del tono bisogna programmare il PIT.

Gli indirizzi I/O del PIT sono:

Indirizzi del PIT

40h = porta dati contatore 0
41h = porta dati contatore 1
42h = porta dati contatore 2
43h = porta di controllo

+ 5 V
8253 PIT
contatore 0
contatore 1
contatore 2
al controller degli interrupt
al circuito di rinfresco memoria
AND
al circuito altoparlante
PPI bit 0
2,38 MHz
÷ 2
PPI bit 1

Gli 8 bit della porta di controllo hanno il seguente formato:

Nel nostro caso scriviamo nella porta di controllo il valore B6h:

Esempio

B6h = 10 11 011 0 = seleziona contatore 2,
prima il byte meno significativo,
poi il più significativo,
modo 3 (generatore di onda quadra),
contatore tipo binary.

Nel modo 3 il contatore opera come un generatore di onda quadra, con frequenza di uscita in conformità alla formula:

frequenza = 1193180 Hz / divisore.

Alla porta dati contatore 2 dobbiamo scrivere il byte meno significativo del divisore seguito dal byte più significativo. Perciò, per una frequenza di 1000 Hz, A9h e successivamente 04h.

Istruzione JMP

NOTA 1: Dopo ogni byte scritto al PIT è necessario aspettare brevamente prima di scrivere un altro byte. Per questo ritardo inseriamo l'istruzione:

```
JMP SHORT $+1
```

la quale segnala al MASM un salto all'istruzione successiva il JMP. Non abbiamo usato l'istruzione NOP perchè il suo ritardo è troppo breve per i processori INTEL più avanzati, ad esempio l'80286.
Naturalmente, questo ritardo non è necessario quando il processore deve eseguire altre istruzioni prima di scrivere al PIT.

Tono modulato

NOTA 2: Il contatore 0 del PIT genera un interrupt 18.2 volte al secondo che interrompe un programma e può causare una modulazione del tono.
Esiste la possibilità di inserire le linee di codice che generanno il suono tra le istruzioni CLI (disabilita interrupt) e STI (abilita interrupt) per avere un tono più costante.

CAPITOLO 9

BIOS

INTRODUZIONE

Read only memory

Per far funzionare un computer e mantenerlo attivo è necessario il software. Se una parte di esso è installato permanentemente il lavoro ne risulta semplificato. A questo scopo viene usata la ROM (READ ONLY MEMORY), cioè una memoria a sola lettura che contiene i programmi e i dati necessari all'avviamento e al funzionamento del computer e delle relative unità periferiche.

Il BIOS (basic input/output system), che è installato in ROM, manipola tutte le istruzioni di un sistema ed è un insieme di routine, richiamabili con un'interruzione, scritte in linguaggio-macchina che forniscono servizi di supporto per il funzionamento del computer. In questo capitolo verranno descritte le principali interruzioni del BIOS. Un'interruzione è un operazione che interrompe l'esecuzione di un programma di modo che il sistema possa intraprendere delle azioni speciali. Le due principali ragioni perchè ciò avvenga sono: una richiesta intenzionale di tale azione, ad esempio, un input da un dispositivo o un output ad un dispositivo e un errore involontario per esempio un overflow.

SERVIZIO DI INTERRUZIONE

Sul PC, la ROM risiede iniziando dalla locazione FFFF0H. Quando viene accesa la macchina, il processore effettua uno stato di reset, esegue un controllo di parità e assegna al registro CS il valore FFFFH e al registro IP il valore zero.

BIOS

La prima istruzione da eseguire è quindi in FFFF:0, o FFFF0, punto di entrata del BIOS. Il BIOS controlla le varie porte di un computer che determinano e inizializzano i dispositivi che sono loro connessi. L'INTEL 8088/8086 riserva, iniziando dalla locazione 0 della memoria, una serie di locazioni di memoria, chiamate vettori di interrupt che contengono gli indirizzi per interrompere quello che occorre. Il BIOS esegue due operazioni che sono INT 11H (determinazione attrezzature) e INT 12H (determinazione capacità memoria). Il BIOS determina inoltre se un dischetto presente contiene il DOS, e, se così, esegue INT 19H per

accedere al primo settore del disco che contenga il bootstrap caricato.

File di sistema

Questo programma legge i file IBMBIO.COM, IBMDOS.COM e COMMAND.COM dal disco.

A questo punto, le memorie RAM e ROM appaiono come segue:

Struttura delle memorie

Vettori di interruzione HARDWARE
Vettori di interruzione del BIOS
Vettori di interruzione del DOS
Vettori di interruzione assegnabili
Area dati ROM-BIOS e DOS
IBMBIO.COM e IBMDOS.COM
Porzione residente del COMMAND.COM
Disponibile per l'uso
Porzione transitoria del COMMAND.COM
Fine RAM
ROM BASIC
ROM BIOS

Interruzione

I dispositivi esterni segnalano le richieste attraverso un interrupt collegato al processore. Il processore risponde ad una richiesta riguardante la disponibilità dell'interruzione (IF) del flag (uguale ad 1) e, nelle migliori circostanze, ignora un'interruzione se l'IF del flag non è disponibile (eguale a 0). L'operazione di interruzione di un'istruzione come può essere INT 12H contiene un tipo di interruzione che ne identifica la richiesta. Per ciascun tipo di operazione, il sistema conserva un indirizzo nei vettori di interrupt iniziando dal settore 0000. Poichè ci sono 256 entrate ogni 4 byte di lunghezza, i vettori occupano i primi 1024 byte di memoria, dall'esa 0 fino all'esa 3FF. Ciascun indirizzo rivela una routine che gestisce uno specifico tipo di interruzione e che consiste in un indirizzo equivalente di codice segmento, il quale inserisce l'interruzione rispettivamente nei registri CS e IP.

Questa struttura fa si che sia possibile richiedere ad un programma un

servizio, senza conoscere la specifica posizione nella memoria della routine di servizio ROM-BIOS.

Permette inoltre di spostare, estendere o adattare i servizi senza che questo influisca sui programmi che li usano.

Le interruzioni disponibili si dividono in tre categorie:

1 - Interruzioni di SERVIZIO dell' 8088/8086

Interruzioni di servizio

00	divide per 0
01	passo singolo (usato dal DEBUG)
02	interrupt non mascherabile (NMI)
03	istruzione di break point (usato dal DEBUG)
04	overflow
05	riservato dalla INTEL (ma usato per stampare il contenuto del video)
06	riservato dalla INTEL
07	riservato dalla INTEL

2 - Interruzioni HARDWARE Pc

Interruzioni hardware

08	8253 real-time-clock
09	interrupt della tastiera
0A	riservato dalla INTEL
0B	riservato dalla INTEL
0C	riservato dalla INTEL
0D	riservato dalla INTEL
0E	interrupt del floppy disk
0F	controller stampante

3 - Interruzioni del BIOS

Verranno elencate e descritte in una sezione successiva.

AREA DATI ROM - BIOS E DOS

Routine di servizio

Ci sono due aree adiacenti di 256 byte che partono dagli indirizzi esa 400 e 500 che controllano molte delle attività del PC. La maggior parte delle posizioni di memoria che le compongono, contengono dati importantissimi per le varie routine di servizio del BIOS e del DOS.

A questi dati possiamo accedere direttamente o tramite le chiamate a un interrupt del BIOS che ci restituisca le informazioni conservate in una particolare locazione.

Segue adesso l'elenco degli indirizzi più utili e del loro contenuto:

indirizzo	0000:0000	interruzioni
indirizzo	0030:0000	stack del BIOS
indirizzo	0040:0000	dati BIOS

Indirizzi fondamentali

0000	dw	indirizzo porta seriale 1
0002	dw	indirizzo porta seriale 2
0008	dw	indirizzo porta parallela 1
0010	dw	dati relativi alle apparecchiature collegate
0012	db	grandezza memoria usabile in K-byte 256=0100, 512=0200, 640=0280
0017	db	byte 1 di stato tastiera
0018	db	byte 2 di stato tastiera
001A	dw	punta all'inizio del buffer della tastiera
001C	dw	punta alla fine del buffer della tastiera
001E	dw	16 word del buffer della tastiera
indirizzo	0050:0000	dati DOS

Indirizzi riservati al DOS

0004	db	indica se il lettore sta funzionando da lettore A (00) o B (01)
0010 : 0021		usati dal BASIC
0100		fine area dati del DOS

Ci pare doveroso avvertirvi che, se scrivete dei programmi che si avvalgono delle informazioni dipendenti dall'hardware o dal BIOS della macchina, essi potrebbero incontrare dei seri problemi quando vengono fatti "girare" sotto DOS su macchine non compatibili IBM e causare gravi errori o cadute del sistema in ambiente multitasking.

LE INTERRUZIONI DEL BIOS

I servizi ROM-BIOS non vengono gestiti da un interrupt principale, ma sono divisi in categorie ciascuna delle quali ha un proprio interrupt. Questo ci permette di sostituire qualsiasi gestione di interrupt riducendo al minimo i problemi.

Sono undici le principali interruzioni del ROM - BIOS divise in cin-

que gruppi:

Gruppi delle interruzioni

- Due agiscono sull'apparecchiatura;
- Sei gestiscono dispositivi periferici specifici;
- Uno gestisce l'orologio dell'ora e della data;
- Uno attiva il ROM-BASIC;
- Uno gestisce l'avvio del sistema;

Nella seguente tabella vengono elencate le principali interruzioni del BIOS.

Numero e significato delle interruzioni

Interrupt Numero servizio in ESA	**APPLICAZIONI**
10	Visualizzazione video
11	Elenco apparecchiature
12	Dimensione memoria
13	I/O Floppy disk e hard disk
14	Comunicazione
15	I/O cassette e funzioni AT
16	Tastiera
17	Stampante
18	Attiva il BASIC da ROM
19	Attiva routine di boot-strap
1A	Ora e data
Altre interruzioni:	
1B	prende contr. sul keyboard-break
1C	prende controllo del timer-inter.
1D	puntatore tabella inizial. video
1E	puntatore tabella param. dischet.
1F	puntatore tabella caratt. grafici

INT 10H VIDEO DISPLAY I/O: Si occupa delle operazioni di interruzione del video e della tastiera, esso verrà descritto meglio in seguito.

INT 11H EQUIPMENT DETERMINATION: Determina i dispositvi opzionali del sistema e rimette un valore in AX. All'accensione, il siste-

ma esegue questa operazione e registra AX nella locazione esa 410.

Significato degli interrupt

INT 12H MEMORY SIZE DETERMINATION: Riporta il formato della memoria in AX in termini di 1K byte tale che 512K di memoria siano esa 200, come determinato durante l'accensione.

Questa operazione è utilizzata per adattare il formato di un programma con la disponibilità della memoria.

INT 13H DISK INPUT/OUTPUT: Provvede agli input/output per i dischetti e l'hard disk.

INT 14H COMMUNICATIONS INPUT/OUTPUT: Fornisce un flusso di byte di I/O alla porta di comunicazione RS232. Il DX conterrà il numero (0 o 1) della porta RS232.

Ci sono quattro servizi disponibili, numerati da 0 a 3, possono essere selezionati tramite il registro AH e vengono usati per inizializzare i parametri della porta seriale, inviare o ricevere un carattere ed infine per acquisire lo stato della porta stessa.

Significato degli interrupt

INT 15H CASSETTE I/O AND AT ADVANCED FEATURES: Provvede alle operazioni di I/O delle cassette e alle operazioni AT. Anche per le cassette sono disponibili 4 servizi, numerati da 0 a 3 e selezionabili tramite il registro AH, con cui possiamo accendere o spegnere il motore e leggere o scrivere blocchi di dati del mangianastri.

INT 16H KEYBOARD INPUT: Provvede ai tre comandi di input della tastiera.

INT 17H PRINTER OUTPUT: Provvede alla stampante.

INT 18H ROM BASIC ENTRY: Chiama il BASIC da ROM.

INT 19H BOOTSTRAP LOADER: Se un dischetto è disponibile, questa operazione legge la traccia 0 nel settore 1 della locazione boot in memoria, sezione 0 che equivale a 7C00 e trasferisce il controllo a questa locazione.

Se non ci sono dischi guida, l'operazione trasferisce al ROM BASIC il punto di entrata via INT 18H. E' possibile usare questa operazione come un'interruzione software; essa non cancella i contenuti del video e non inizializza i dati in ROM BIOS.

INT 1AH TIME OF DAY: Sistema e legge l'ora rimettendo un valo-

re in AH. Attraverso il suo utilizzo è possibile determinare, ad esempio, il tempo di esecuzione di una routine, sistemando l'orologio a zero, quindi leggendolo alla fine del processo.

Convenzioni operative

Per gestire i servizi del ROM-BIOS ci si serve di alcune convenzioni operative comuni per la loro chiamata, ciò per rendere omogeneo e coerente l'uso dei registri, stack e memoria.

Il registro CS (code segment) è riservato, caricato e ripristinato in modo automatico, per cui nei programmi da noi implementati non dobbiamo occuparcene.

Noi consigliamo in generale di far lavorare i programmi con uno stack più grande di quello necessario ai servizi ROM-BIOS che è di circa 20 byte.

I registri DS e ES (data segment, extra segment) se non vengono usati, sono conservati dalla routine dei servizi del ROM-BIOS.

Registri

I registri AX e DX sono disponibili quando il loro intervento durante la chiamata di un servizio non è fondamentale, per esempio, quando si ha un passaggio di parametri. I registri DI e SI sono disponibili.

OPERAZIONI SU DISCO COL BIOS

Input/Output

In questa sezione tratteremo i servizi per la gestione delle operazioni di I/O di floppy-disk e hard-disk. Vi consigliamo di usare questi servizi con molta prudenza, solitamente è preferibile lasciarne la gestione al sistema operativo.

Potete codificare direttamente dal livello BIOS il procedimento per il disco, con il BIOS potete fornire un uso non automatico della directory o un bloccaggio e uno sbloccaggio dei registri.

L'operazione su disco BIOS INT 13H tratta tutti i "record" come il formato di un settore e indirizza il disco in termini di reale numero traccia e numero settore.

Per leggere, scrivere, e verificare i dischi, iniziate dai seguenti registri:

Registri di lettura e scrittura

AH	Tipo operazione: leggere, scrivere, verificare, o formattare.
AL	Numero dei settori.
CH	Tracce numeriche.
CL	Inizio numero settore.

DH Intesta i numeri: 1 o 0 per i dischetti.
DL Numero guida: 0 = Drive A, 1 = drive B e così via.
ES:BX Iindirizzo del buffer di I/O nell'area dati (eccetto che per le operazioni di verifica).

Il BIOS INT13H richiede un codice nel registro AH per identificare l'operazione.

Applicazioni

Servizio	Applicazioni
0	Resetta il sistema dischetti
1	Acquisisce lo stato del dischetto
2	Legge settori da dischetto
3	Registra settori su dischetto
4	Verifica settori su dischetto

AH = 00 SISTEMA DI RESET DEL DISCHETTO

Questa operazione esegue un reset del controller del dischetto e del lettore, e richiede solamente il valore 00 in AH per eseguire INT 13H.

Questa procedura viene usata solitamente dopo un'operazione su disco che riporta un grave errore.

AH = 01 LEGGE LO STATO DEL DISCHETTO

Funzioni operative

Questa operazione riporta dal dischetto in AL la situazione dell'ultima operazione di I/O. L'esecuzione richiede solamente il valore esadecimale 01 nel registro AH.

Conservando lo stato del dischetto, una routine che gestisca o riporti gli errori può essere molto utile.

AH = 02 LEGGE I SETTORI DA DISCHETTO

L'operazione legge, nella memoria, uno specifico numero di settori sulla stessa traccia. Il numero può essere 1, 8, o 9.
L'operazione carica il registro BX con l'indirizzo della memoria per l'area di input, da notare che BX in questo caso è assoggettato ad ES, da questo la forma ES:BX.

Al ritorno, AL contiene il numero di settori che l'operazione realmente legge. I registri DS, BX, CX e DX sono protetti.

Per una migliore situazione, un programma dovrebbe specificare solamente un settore o tutti i settori per ogni traccia. Esso dovrebbe inizializzare CH e CL e dovrebbe incrementarli per leggere in modo sequenziale i settori.

Funzioni operative

AH = 03 SCRIVE I SETTORI SU DISCHETTO

Questa operazione trascrive una specifica area dalla memoria (presumibilmente 512 byte o un multiplo di 512) in un designato settore. Carica i registri e maneggia i processi proprio come per leggere un disco (cod. 02).

Al ritorno, AL contiene il numero di settori che tramite l'operazione sono stati realmente scritti. I registri DS, BX, CX e DX sono protetti.

AH = 04 VERIFICA SETTORI

Questa operazione controlla semplicemente che gli specifici settori possano essere trovati ed esegue un tipo di controllo di parità.

Collocate i registri proprio come per il codice 03, ma fino a quando l'operazione non ha eseguito una vera e propria verifica, non c'è bisogno di porre un indirizzo in ES:BX. Al ritorno, AL contiene il numero di settori realmente verificati. I registri DS, BX, CX e DX sono protetti.

AH = 05 FORMATTAZIONE TRACCE

Usate questa operazione per formattare un certo numero di tracce in conformità ad uno dei quattro formati (lo standard per il sistema PC è 512).

Le operazioni di lettura e scrittura richiedono un formato di informazioni da inserire in un settore richiesto.

Funzioni operative

Per questa operazione, i registri ES:BX devono contenere un indirizzo che punti ad un gruppo di campi indirizzati per la traccia.

Per ciascun settore sulla traccia, ci devono essere quattro byte di entrata nella forma T/H/S/B, dove:

T = numero traccia
H = numero testina
S = numero settore
B = byte per settore (00=128, 01=256, 02=512, 03=1024)

STATO DEL BYTE

Per tutti i codici AH 02, 03, 04 e 05, se l'operazione ha buon esito, il flag CF e AH sono collocati a 0. Se l'operazione fallisce, il flag CF è collocato a 1 e AH contiene un codice di stato che identifica la causa del fallimento.

Nella seguente tabella riportiamo le possibili configurazioni del byte di stato:

VALORE (ESA) IN AH	CORRISPONDENTE BINARIO 76543210	DESCRIZIONE
1	00000001	Comando errato: è pervenuta al controller una richiesta non valida.
2	00000010	La marca che identifica il settore non è valida o non è presente.
3	00000011	Tentata scrittura su un disco protetto.
4	00000100	Il settore richiesto non è presente sul disco.
8	00001000	Errore nel DMA.Cattivo funzionamento.
9	00001001	Tentata una DMA al di fuori dell'area di 64 K.
10	00010000	Rilevato un errore di parità sui dati dalla lettura del dischetto. CRC danneggiato.
20	00100000	Funzionamento anomalo del controller del dischetto.
40	01000000	La ricerca della traccia non è andata a buon fine.
80	10000000	Time out. Nessun responso dal lettore.

Se all'interruzione dell'operazione ricompare un errore, si consiglia di resettare il controller del disco (AH codice 00) e riprovare l'operazione tre volte. In presenza di un errore, compare un messaggio che da all'utente la possibilità di poter cambiare il dischetto.

STAMPARE USANDO IL BIOS INT 17H

I servizi ROM-BIOS per la stampante supportano l'output della stessa.

Il comando BIOS 17H effettua tre differenti operazioni specificate nel registro AH:

AH = 0 Questa operazione provoca la stampa di un carattere e consente l'uso di tre stampanti, numerate 0, 1 e 2 (0 è il valore di default).

Istruzioni di stampa

```
MOV AH,00          Richiesta di stampa
MOV AL,carat       Carattere da stampare
MOV DX,01          Seleziona la stampante 1
INT 17H            Chiama il BIOS
```

Se l'operazione di stampa del carattere fallisce, il registro AH è posto a 01.

AH = 1 Inizializza la porta della stampante come segue:

Inizializzazione

```
MOV AH,01          Richiede l'inizializzazione della porta
MOV DX,00          Seleziona la porta della stampante 0
INT 17H            Chiama il BIOS
```

Poichè l'operazione spedisce un carattere di form feed, usatelo per posizionare la stampante alla posizione più alta della pagina sebbene le migliori stampanti lo fanno già automaticamente.

AH = 2 Legge lo stato della porta della stampante come segue:

Stato della stampante

```
MOV AH,02          Richiede la lettura della porta
MOV DX,00          Seleziona la porta della stampante 0
INT 17H            Chiama il BIOS
TEST AH,00101001B  Pronta?
JNZ msgerr         No -- visualizza messaggio di errore
```

Il proposito di AH = 1 e AH = 2 è quello di determinare lo stato della stampante.

L'operazione posiziona i bit in AH a 1 come segue:

BIT	CAUSA
7	Stampante raggiungibile
6	Riconoscimento stampante disponibile
5	Fine della carta
4	Stampante selezionata
3	Errore di I/O
0	Time out

Messaggi d'errore

E' consigliabile prima di un'operazione di stampa che usa il BIOS, controllare lo stato della porta; se c'è un errore, viene visualizzato un messaggio (L'operazione DOS si occupa di questo controllo automaticamente, attraverso il suo messaggio "fine carta" applicato a varie condizioni).

Ad ogni modo, una stampante può andare fuori formato o può essere inavvertitamente disattivata. Conseguentemente, se volete scrivere un programma usato anche da altri, includetegli uno stato di test prima di ogni tentativo di stampa.

I SERVIZI VIDEO DEL ROM-BIOS

In questa sezione ci occuperemo delle più avanzate caratteristiche relative al video e alle possibili azioni che vi si possono svolgere, come lo scorrimento del video o la collocazione di un byte di attributo per sottolineature, lampeggiamenti e alta intensità

Video monocromatico

Memoria video

Il video monocromatico supporta 4 K byte di memoria (display buffer) iniziando dall'indirizzo esa B0000. Questa memoria comprende:

2K byte di 25 righe e 80 colonne di caratteri.
2K byte per configurare i caratteri di attributo del video: il reverse video,

lampeggiamento, alta intensità e sottolineatura.

Video a colori e grafico

Grafica

Il video standard a colori/grafico supporta 16K byte di memoria (display buffer) iniziando dall'indirizzo esa B8000. Questo video può operare o a colori o in bianco e nero e ha dei modi per visualizzare testi (normali caratteri ASCII) e grafici. Il buffer del display fornisce la numerazione alle pagine video da 0 a 3 per 80 colonne video e da 0 a 7 per 40 colonne video. Il numero della pagina di default è zero, ma potete formattare qualunque pagina in memoria.

BYTE DEGLI ATTRIBUTI

Modo testo

Il byte di attributo per entrambi i video a colore e monocromatico, in modo testo (non grafico), determina le caratteristiche di ciascun carattere visualizzato. Il byte degli attributi gestisce le seguenti caratteristiche:

	BACKGROUND				FOREGROUND			
ATTRIBUTO:	BL	R	V	B	I	R	V	B
NUM. BIT :	7	6	5	4	3	2	1	0

Testo e sfondo

Ciascuna lettera RVB rappresenta una posizione di bit che significa, per un monitor a colori, rosso, verde e blu. Il bit 7 (BL) attiva il lampeggiamento, e il bit 3 (I) attiva l'alta intensità. Su un monitor monocromatico, il foreground è verde o ambrato e il background è nero, questa tabella fa riferimento al video in bianco e nero. Per modificare questi attributi, potete combinarli come segue:

FUNZIONE	BACKGROUND RVB	FOREGROUND RVB
Nondisplay (nero su nero)	000	000
Sottolinea (no per colore)	000	001
Video normale: bianco su nero	000	111
Video opposto: nero su bianco	111	000

I SERVIZI DEL BIOS INT 10H

Sotto questo interrupt ci sono sedici servizi principali e un servizio AT. I parametri addizionali vengono di solito inseriti in BX, CX, o DX.

SERVIZI (valori espressi in ESA)	**APPLICAZIONI**
0	Determina il video mode.
1	Determina la dimensione del cursore.
2	Determina la posizione del cursore.
3	Legge la posizione del cursore.
4	Legge la posizione della penna luminosa.
5	Determina la pag. attiva di visualizzaz.
6	Scorre la finestra verso l'alto.
7	Scorre la finestra verso il basso.
8	Legge il carattere e il suo sottoattributo.
9	Scrive il carattere e il suo sottoattributo.
A	Scrive il carattere.
B	Determina la tavola colori da usare.
C	Scrive un punto pixel.
D	Legge un punto pixel.
E	Scrive i caratteri in TTY mode.
F	Legge il videomode corrente, ampiezza schermo e numero pagina.
13	Scrive sullo schermo una stringa di caratteri (disponibile solo su AT).

Interrupt video

L'INT 10H facilita il pieno controllo del video. Inserite nel registro AH un codice che determini la funzione del comando. L'operazione preserva i contenuti dei registri BX, CX, DX, DI, SI, e BP. La tabella elenca i diciassette servizi video che verranno successivamente descritti:

AH = 00 SET MODE

Usate questa operazione per muovervi tra i modi testo e grafico. Usate INT 10H per fissare il modo per la corrente esecuzione del programma. Quasi tutti i PC-IBM e compatibili hanno un adattatore video che supporta i modi del monochrome adaptor (MA) e il color graphics adaptor (CGA):

MODO	TIPO	DIMENSIONI	ADATTATORE
0	TESTO b/n	40 x 25	CGA
1	TESTO 16 col.	40 x 25	CGA
2	TESTO b/n	80 x 25	CGA
3	TESTO 16 col.	80 x 25	CGA
4	GRAFICA 4 col.	320 x 200	CGA
5	GRAFICA 4 grigi	320 x 200	CGA
6	GRAFICA b/n	640 x 200	CGA
7	TESTO b/n	80 x 25	MA
Seguono molti altri modi per adattatori video avanzati come, ad esempio: PCjr, EGA, MCGA, VGA, HERCULES, COMPAQ, OLIVETTI, etc.			

Il seguente esempio fissa il video mode per testo standard bianco e nero:

Esempio

```
MOV AH,00      Richiesta di collocazione modo
MOV AL,02      Testo standard bianco e nero 80 x 25
INT 10H        Chiama il BIOS.
```

Se scrivete del software per video monitor sconosciuti, potete usare BIOS INT 11H per determinare il dispositivo accluso al sistema; l'operazione rimette un valore ad AX, con i bit 5 e 4 indicanti il videomode:

Videomode

01	State usando un adattatore a colori 40 x 25
10	State usando un adattatore a colori 80 x 25
11	State usando un adattatore bianco e nero 80 x 25

Esaminate AX per il tipo di monitor e poi fissate il modo. Quando si fissa il modo il ROM-BIOS svuota la memoria di transito dello schermo, anche se il modo era lo stesso, può quindi essere usato molto facilmente per svuotare lo schermo.

AH = 01 STABILISCE LA DIMENSIONE DEL CURSORE

Istruzioni per il video

Il cursore non fa parte dei caratteri ASCII. Il computer mantiene il proprio hardware per il suo controllo e vi sono delle speciali operazioni INT per questo uso. Il normale simbolo del cursore è simile ad una sottolineatura o ad un carattere di spazio. Usate INT 10H per fissare la sua forma verticalmente: fissate CH (bit 4 -0) per il punto più alto e CL (bit 4-0) per quello inferiore.

Potete modificare il formato tra la parte più alta e la parte inferiore, 0/13 per il video monocromatico e enhanced graphics, e 0/7 per i principali video a colori. Ad esempio, per estendere il cursore dalla sua massima alla sua minima posizione:

```
MOV    AH,01      Richiesta modifica formato cursore
MOV    CH,00      Prima linea di scansione
MOV    CL,13      Ultima linea di scansione (mono)
INT    10H        Chiama il BIOS
```

Il cursore ora lampeggia come un rettangolo solido. Usando 12/13 (mono) o 6/7 (colore) risettate il cursore normale. Per togliere il cursore nei modi testo una delle tecniche consiste nel fissare a 1 il bit 5 del registro CH.

AH = 02 STABILISCE LA POSIZIONE DEL CURSORE

Cursore

Questa operazione pone il cursore in un qualsiasi punto del video in accordo con le coordinate delle righe e delle colonne. Il numero della pagina è normalmente 0, ma per il modo a 80 colonne può essere da 0 a 3 e da 0 a 7 per il modo a 40 colonne. Ponete in AH 02, in BH il numero della pagina, e in DX la riga colonna:

```
MOV    AH,02      Richiesta di movimento cursore
MOV    BH,00      Pagina 0
MOV    DH,riga    Riga
MOV    DL,col     Colonna
INT    10H        Chiama il BIOS
```

AH = 03 LEGGE LA POSIZIONE CORRENTE DEL CURSORE

Un programma può determinare la riga, colonna e il formato del cursore corrente come segue:

Posizione cursore

```
MOV    AH,03      Richiesta posizione cursore
MOV    BH,00      Specifica il numero pagina
INT    10H        Chiama il BIOS
```

Al ritorno, DH contiene la riga, DL la colonna, CH la prima linea di scansione e CL l'ultima linea di scansione.

AH = 04 LEGGE LA POSIZIONE DELLA PENNA LUMINOSA

Determina lo stato della penna luminosa: se è stata attivata e, in tal caso, dove si trova sullo schermo.

AH = 05 SELEZIONA, ATTIVA UNA PAGINA

Selezione pagina

Per i modi di testo, selezionare una nuova pagina da 0 a 3. Nel 40 colonne, le pagine possono da essere 0 a 7; nell'80 colonne, 0 a 3. La pagina 0 si trova all'inizio della memoria dello schermo, quelle successive dopo 2K nei modi 40 colonne o dopo 4K nei modi 80 colonne.

```
MOV    AH,05        Attiva una pagina sullo schermo
MOV    AL,pagina    Numero pagina
INT    10H          Chiama il BIOS
```

AH = 06 SCORRI IN ALTO LO SCHERMO

Questo servizio è usato per definire un'area dello schermo, che noi possiamo chiamare finestra, per poi farne scorrere il contenuto di una o più righe in alto. Il numero di righe che dovrà scorrere, deve essere specificato in AL. Per il formato testo, collocando 00 in AL si provoca uno scorrimento in alto dell'intero video e tutta la finestra viene svuotata.

I dati relativi alla finestra che dovrà scorrere devono essere specificati nei registri: DX (DH = ultima riga, DL = colonna di destra) e BH = attributo schermo delle nuove righe.

AH = 07 SCORRI IN BASSO LO SCHERMO

Scrolling

Questo servizio è usato per definire un'area dello schermo, che noi possiamo chiamare finestra, per poi farne scorrere il contenuto di una o più righe in basso.

Una volta inizializzato AH con il valore 7, l'uso degli altri registri è esattamente identico a quello già descritto per lo scorrimento in alto dello schermo.

AH = 08 LEGGE L'ATTRIBUTO E IL CARATTERE NELLA CORRENTE POSIZIONE DEL CURSORE

Usate le seguenti istruzioni per leggere sia i caratteri sia gli attributi dall'area video, nel formato testo o grafico:

Caratteri

```
MOV    AH,08        Richiesta lettura attributo/carattere
MOV    BH,00        Pagina 0 (per il formato testo)
INT    10H          Chiama il BIOS
```

L'operazione rimette il carattere in AL e il suo attributo in AH. In formato grafico, l'operazione rimette il valore 00 per un carattere non ASCII. La pagina nel modo di testo deve essere specificata in BH. Nel modo grafico non è necessario fissarla.

AH = 09 SCRIVE L'ATTRIBUTO E IL CARATTERE NELLA CORRENTE POSIZIONE DEL CURSORE

Questa operazione visualizza i caratteri in formato testo o grafico con blanking, video opposto, ecc.

Posizione carattere

Il valore posto in AL è un singolo carattere che può essere visualizzato diverse volte. Il valore in CX determina il numero di volte che il carattere posto in AL deve essere visualizzato. L'operazione visualizza i diversi caratteri richiesti a loop. L'operazione non porta avanti automaticamente il cursore.

Nel modo di testo e non in quello grafico, il carattere duplicato scorre automaticamente dalla parte destra del video alla linea successiva. Se si vuole fare una sola copia del carattere, è necessario assicurarsi di porre a 1 il registro CX. Per il formato grafico, usate BL per definire il foreground colore. Se il bit 7 è 0, il colore definito rimpiazza l'attuale pixel colore; se il bit 7 è 1, il colore definito è combinato (XOR) con quello attuale.

AH = 0A SCRIVE UN CARATTERE ALL'ATTUALE POSIZIONE DEL CURSORE

Colore

La sola differenza tra il codice 0A e 09 è che 0A in modo di testo non permette di cambiare l'attributo del colore dallo schermo. Tuttavia è necessario, in modo grafico, specificare il colore in BL.

AH = 0B DETERMINA LA TAVOLA COLORI DA USARE

Tabella colori

Questo servizio viene usato quando si deve scegliere una delle due tavolozze grafiche per la media risoluzione (0 o 1 nella grafica da 320 x 200). L'identificativo della tavola di colore si carica in BH mentre il colore o valore della tavola da usare con quell'identificativo si carica il BL.

AH = 0C SCRIVE UN PUNTO PIXEL

Questo sevizio scrive un pixel singolo. I registri usati per definire la posizione del pixel sono:

Pixel

DL per il numero di riga (1 byte);
CX per il numero di colonna (2 byte).

Il colore va posto in AL (vedi servizio 9 per la scelta colore diretto o trattato XOR).

AH = 0D LEGGE UN PUNTO PIXEL

La chiamata a questo servizio restituisce in AL il codice di colore del pixel. Come per il servizio 0C la riga va specificata in DL e la colonna in CX.

AH = 0E SCRIVE UN CARATTERE IN MODO TTY

Comunicazione TTY

Questa operazione Vi permette di usare un monitor come un semplice terminale. Collocate in AH il valore esa 0E, il carattere da visualizzare in AL, il colore di foreground (formato grafico) in BL e il numero della pagina (formato testo) in BH. Cicalino (07H), spazio indietro (08), introduci una linea (0AH), e ritorno carello (0DH) agiscono come comandi per formattare il video.
L'operazione fa avanzare automaticamente il cursore, sovrappone i caratteri sulla nuova linea, fa scorrere il video, e conserva i presenti attributi del video. Nel Modo Testo, gli attributi attuali vengono mantenuti da un carattere al successivo. Ciò non avviene nel colore dove, il colore di foregroud (di primo piano) deve ogni volta essere specificato nel registro BL.

AH = 0F DA IL MODO CORRENTE DEL VIDEO

Questa operazione rimette il modo del video (vedi codice AH = 00) in AL, l'ampiezza dello schermo (caratteri per linea) in AH (20, 40, o 80) e il numero della pagina di visualizzazione in BH.

AH = 13 VISUALIZZA UNA STRINGA DI CARATTERI (DISPONIBILE SOLAMENTE SU AT)

Stringhe

Questa operazione permette agli utenti dell'AT di visualizzare delle stringhe con delle opzioni di collocamento e movimento del cursore.

Si può lasciare il cursore fermo o spostarlo a seconda del sotto-servizio che si sceglie. Il numero di sotto - servizio si mette in AL.

I quattro sotto-servizi sono:

0 Usa l'attributo e non muove il cursore in avanti

1 Usa l'attributo e fa avanzare il cursore
2 Visualizza il primo carattere, l'attributo e non fa avanzare il cursore
3 Visualizza il primo carattere, poi l'attributo e fa avanzare il cursore.

Il puntatore alla stringa va posto in ES:BP; la posizione di inizio in cui scrivere la stringa in DX; la lunghezza della stringa va posta in CX; mentre i registri BH e BL contengono rispettivamente il numero di pagina di visualizzazione e l'attributo.

I SERVIZI DELLA TASTIERA CON IL BIOS INT 16H

I servizi a cui il ROM-BIOS INT 16H provvede sono tre; essi sono numerati da 0 a 2 e vengono selezionati tramite il registro AH.

SERVIZI	APPLICAZIONE
0	Legge il successivo carattere in input dalla tastiera.
1	Avvisa se un carattere è disponibile.
2	Acquisisce lo stato del tasto shift.

AH = 00 LEGGE IL PROSSIMO CARATTERE

L'operazione legge in AL il successivo carattere ASCII entrato dalla tastiera e colloca il codice di scansione standard in AH.

Per i caratteri speciali ritorna 0 in AL e l'indicativo del carattere speciale in AH.

Tasti funzione

Se il carattere immesso è un carattere speciale come Home, o un tasto funzione AL è posto a 00.

Se non c'è nessun carattere in attesa nella memoria di transito della tastiera, il servizio aspetta finchè non ce n' è uno, sospendendo in pratica le operazioni di elaborazione successive del programma.

L'operazione non riporta automaticamente l'eco del carattere sul video.

AH = 01 RIPORTA SE UN CARATTERE E' DISPONIBILE

Viene usato come segnale il flag zero: indica che un carattere è disponibile per la lettura.

Lettura caratteri

Il prossimo carattere e il codice di scansione ad essere letti, sono in AL e AH, rispettivamente, mentre l'input rimane nel buffer.

Il servizio 1 è particolarmente utile in fase di programmazione quando, per esempio, si vuole svuotare la memoria di transito della tastiera per impedire che qualsiasi input possa intervenire in una fase non voluta o, per esempio, quando si vuole interrompere un processo di attesa di un segnale dalla tastiera.

AH = 02 RITORNA LO STATO DEL TASTO SHIFT

Tasto shift

L'operazione preleva lo stato del tasto shift della tastiera contenuto alla locazione esadecimale 417 della memoria e lo pone in AL.

La seguente tabella mostra i bit di stato della tastiera che possono essere restituiti:

BIT								DESCRIZIONE
7	6	5	4	3	2	1	0	
1								Inserimento attivo
	1							Caps Lock attivo
		1						Bloc Num attivo
			1					Bloc Scorr attivo
				1				Alt shift attivo
					1			Ctrl shift attivo
						1		Tasto shift sinistro attivo
							1	Tasto shift destro attivo

Il BIOS INT 16H non agisce direttamente sull' hardware ma esegue le sue operazioni di lettura o controllo sul buffer della tastiera. Chi agisce direttamente sulla tastiera è la routine di servizio dell'interruzione 09H di cui faremo adesso una brevissima descrizione.

Ogni volta che un tasto viene premuto o rilasciato, il controller della tastiera genera un interrupt 09h. Il programma corrente è sospeso, e la routine di servizio dell'interruzione viene chiamata.

La routine di servizio dell'interrupt 09h interroga il controller della tastiera per sapere quale tasto era stato attivato. Ogni tasto genera un codice di 8 bit quando viene premuto o rilasciato. Successivamente questo codice di scansione viene controllato e si genera un codice ASCII.

Questi due codici vengono posti nella memoria di transito della tastiera.

Codici di scansione

Nel BIOS, la routine di servizio interpreta i codici di scansione in questo modo:

Per tasti speciali, come BREAK, chiama una routine del BIOS.

Per tasti come CTRL, ALT e SHIFT, mette a 0 o a 1 un bit di status nella zona dati del BIOS.

Per il tasto ALT in combinazione con i tasti della tastiera numerica, genera il codice ASCII corrispondente.

Per i tasti normali, usa il codice di scansione più i bit di status per CTRL, ALT e SHIFT per generare un indice nella tavola dei codice ASCII.

Quando un programma ha bisogno di leggere la tastiera usa il DOS o il BIOS per prendere i codici dei tasti dalla memoria di transito. L'interrupt 16h viene usato per la chiamata tramite il BIOS; anche il DOS si serve di questa interruzione.

Quindi, abbiamo tre metodi per leggere la tastiera: Interrupt 09h, interrupt 16h e DOS.

Scrivere una nuova routine di servizio per l'interrupt 09h permette:

Lettura della tastiera

- 1. Il controllo di ogni tasto premuto o rilasciato, compresi i tasti normalmente non disponibili con il BIOS e il DOS. Quasi tutti i tasti generano diversi codici quando sono premuti e quando vengono rilasciati, ma pochi codici generati al rilascio vengono usati dal BIOS o dal DOS.

- 2. Il cambiamento dei codici ASCII e di scansione per tutti i tasti, e perfino la loro sostituzione con una stringa di codici. Per esempio, è possibile usare il tasto 5 sulla tastiera numerica per una funzione speciale.

Zona dati BIOS:

Indirizzi dei dati

0040:0017	db	status CTRL, ALT, SHIFT.
0040:0018	db	status tasti speciali.
0040:001A	dw	puntatore all'inizio della memoria di transito
0040:001C	dw	puntatore alla fine della memoria di transito
0040:001E	dw	memoria di transito (32 word) (16 codici scansione + ASCII)

ESTENSIONE TASTI DI FUNZIONE

La tastiera provvede a tre tipi di tasti base:

- 1- I caratteri alfabetici dalla A alla Z, numerici da 0 a 9, %,$ e così via.

Tipi di tasti

- 2- I tasti di funzione come Home, Fine, Backspace , Frecce, Invio, Canc, Ins, Pagina in alto e in basso e i tasti di funzione di programma.

- 3- Tasti di controllo per Alt, Ctrl e Shift che lavorano in associazione con gli altri tasti.

Non vi è nulla nel disegno del tasto che ci faccia presumere l'esecuzione di una specifica azione. Come programmatori, dovete sapere, per esempio, che premendo il tasto Home si colloca il cursore nell'angolo sinistro in alto del video, o che premendo il tasto Fine si colloca il cursore al termine del testo sul video.

Potrete così programmarli facilmente ad eseguire un'operazione to-

Scelta dei tasti

talmente senza legami. Ciascun tasto ha un codice di scansione designato, numerato da 1 (Esc) fino a 83 (Canc) o da 01 fino a 53 esadecimale. Attraverso il codice di scansione principale, un programma può determinare la sorgente di qualunque tasto premuto.

Per esempio, una richiesta di input di un carattere dalla tastiera richiede 00 nel registro AH e viene fornito dallo interrupt 16H, come segue:

```
MOV AH,00 Richiede input dalla tastiera
INT 16H Chiama il BIOS
```

L'operazione risponde in uno o due modi, dipendentemente se premete un tasto carattere o un tasto funzione. Per un carattere come la lettera C, la tastiera ritorna al computer due informazioni:

Operazioni di base

- Il carattere ASCII C (esa 43) in AL
- Il codice di scansione per la lettera C, esa 20, in AH.

Se premete un tasto funzione come Fine, la tastiera ritorna ugualmente due informazioni:

- Zero nel registro AL
- Il codice di scansione per Fine, esa 4F, in AH.

Inoltre, dopo ogni istruzione INT 16H, potreste controllare AL. Se è

Codici di scansione

FUNZIONE (esadecimale)	CODICE DI SCANSIONE
da ALT A a ALT Z	1E - 2C
da F1 a F10	3B - 44
Home	47
Freccia in alto	48
Pagina in alto	49
Freccia a sinistra	4B
Freccia a destra	4D
Fine	4F
Freccia in basso	50
Pagina in basso	51
Ins	52
Canc	53

a Zero, la richiesta è per un codice funzione; se non è zero, l'operazione ha consegnato un carattere.

La tabella precedente elenca i codici di scansione corrispondenti alle funzioni estese.

CAPITOLO 10

IL SISTEMA OPERATIVO DOS

CRONISTORIA

Il Disk Operating System (MS-DOS o PC-DOS) è il sistema operativo dominante nel mercato dei personal computer che usano i micro-processori Intel 8086-80x86. Ciò è dovuto, in gran parte, alla adozione da parte dell'IBM, del DOS come sistema operativo per i propri PC.

Versioni DOS

Dal punto di vista del programmatore, le versioni correnti del DOS (Versione 2, 3 e 4) sono un ricco e potente ambiente di sviluppo. Una grande scelta di mezzi di programmazione è disponibile dalla Microsoft o da altre software house e il porting delle applicazioni già esistenti con l'ambiente DOS non è difficile.

Il progenitore del DOS era una sistema operativo chiamato 86-DOS, scritto da Tim Paterson per la Seattle Computer Products nel 1980. L'86-DOS divenne il sistema operativo per la linea dei microcomputer S-100 bus, basati sull' Intel 8086, della Seattle Computer Products.

Nell' autunno 1980 l'IBM cominciò a cercare un sistema operativo per il suo nuovo personal computer. A questo punto alla Microsoft non avevano un sistema operativo loro da poter offrire e pagarono alla Seattle Computer Products il diritto per vendere l'86-DOS scritto da Paterson.

MS-DOS

Nell' estate 1981 la Microsoft comprò tutti i diritti sullo 86-DOS e, dopo cambi sostanziali, lo chiamò MS-DOS. Con il primo PC IBM, uscito nell' autunno del 1981, l'IBM offriva l' MS-DOS (PC-DOS 1.0) come sistema operativo principale.

L' IBM fu l'unica, tra i maggiori fabbricanti di computer, a includere l'MS-DOS 1.0 (chiamato PC-DOS 1.0) nei suoi prodotti. L' MS-DOS 1.25 (equivalente a PC-DOS 1.1) fu prodotto nel Giugno 1982 per curare qualche "bug" e per supportare la gestione dei dischetti a doppia faccia. Questa versione migliorava nell' indipendenza dall'hardware il DOS kernel. Ora altri venditori, come Compaq e Texas Instruments offrono il DOS con le loro macchine.

ORGANIZZAZIONE

In questo capitolo discuteremo su come il DOS è organizzato e come viene caricato in memoria quando il computer viene acceso.

La struttura del dos

Il DOS è suddiviso in alcuni strati che servono ad isolare la logica del kernel dal sistema operativo, e a sganciare l'utente dell' hardware su cui sta lavorando. Questi strati sono:

Struttura

- Il BIOS (Basic Input/Output System)
- Il kernel DOS
- Il processore dei comandi (shell)

Il modulo bios

Il BIOS è specifico del sistema e viene fornito dal fabbricante. Contiene i default resident hardware per i seguenti dispositivi:

Console display e tastiera (CON)

Line printer (PRN)

Auxiliary device (AUX)

Dispositivi

La data e l'ora (CLOCK)

Boot disk device (block device)

Il kernel del DOS comunica con queste unità driver attraverso pacchetti di richieste I/O; i driver traducono queste richieste in veri comandi per i differenti controller dell'hardware. In molti sistemi DOS, compreso il PC IBM, le routine che gestiscono le parti più primitive dall'hardware sono locate nella memoria di sola lettura (ROM-Read Only Memory). Così possono essere usate dal programma di boot del sistema, da applicazioni stand alone, da programmi diagnostici, ecc..

CONFIG:SYS

I termini residenti e installabili sono usati per distinguere i driver contenuti dentro il BIOS dai driver installati durante il boot-up del sistema e dai comandi DEVICE del file CONFIG.SYS.

File hidden

Il BIOS viene letto dentro la memoria d'accesso casuale (RAM Random Access Memory) durante l'inizializzazione del sistema come una parte del file chiamato IBMBIO.COM. Questo file ha gli attributi hidden (nascosto) e system (file del sistema).

Il kernel dos

Il kernel è un programma che fornisce un insieme di servizi indipendenti dall'hardware chiamati system functions (funzioni di sistema). Incluse in queste funzioni sono:

Funzioni del kernel

- La gestione di file e record
- La gestione della memoria
- L'unità input/output di caratteri
- Lo "Spawning" di altri programmi
- L'accesso all'orologio real-time

I programmi possono avere accesso alle funzioni di sistema caricando i registri con parametri specifici per la funzione e poi trasferendoli al sistema operativo fra una chiamata o un interruzione software. Il kernel DOS viene letto in memoria durante l'inizializzazione del sistema dal file IBMDOS.COM sul dischetto di boot.

Il processore dei comandi

Interfaccia utente

Il processore dei comandi è l'interfaccia fra l'utente e il sistema operativo, ed è responsabile dell'analisi e dell'esecuzione dei comandi dell'utente, incluso il caricamento e l'esecuzione dei programmi da dischetto o altro apparato per l'archiviazione di massa (mass storage device).

Lo shell di default fornito dal DOS, è contenuto nel file COMMAND.COM. Di solito, i prompt e le risposte fornite dal COMMAND.COM sono l'unica percezione che si ha del DOS ma è importante capire che COMMAND.COM non è il sistema operativo bensì, un tipo di programma speciale che gira sotto il controllo del DOS.

Il COMMAND.COM è diviso in tre parti:

COMMAND COM

- Una porzione residente
- Una sezione d'inizializzazione
- Un modulo transiente

La porzione residente è caricata nella parte più bassa della memoria, sopra il kernel del DOS e i suoi buffer e tabelle. Contiene le procedure per elaborare le sequenze CTRL-C e CTRL-BREAK, gli errori critici e la terminazione (uscita finale) di altri programmi transienti. Questa parte del COMMAND.COM è quella che emette i messaggi di errore. Contiene anche il codice usato per ricaricare la parte transiente quando ciò è necessario.

La sezione d'inizializzazione del COMMAND.COM viene caricata sopra la parte residente quando il sistema è acceso. Questa parte elabora il file batch AUTOEXEC.BAT (se è presente) e successivamente viene scartata.

Il modulo transiente del COMMAND.COM è caricato nella parte più alta della memoria. Questa parte della memoria può anche essere usata per altri scopi dai programmi applicativi. Il modulo transiente emette il prompt dell'utente, legge i comandi dalla tastiera o file batch ed esegue i comandi.

Quando un programma applicativo è terminato, la porzione residente del COMMAND.COM esegue un checksum del modulo transiente per determinare se esiste ancora e, se necessario, ne carica una nuova copia dal disco di boot.

I comandi dell'utente accettati dal COMMAND.COM sono di tre categorie:

Comandi utente

- 1 - Comandi interni
- 2 - Comandi esterni
- 3 - File batch

Comandi interni

I comandi interni, chiamati anche comandi intrinseci hanno le procedure incluse nella parte transiente del COMMAND.COM. Tra i comandi di questa categoria ricordiamo: REN(AME), PATH DIR(ECTORY), MORE, MKDIR and DEL(ETE).

Comandi esterni

I commandi esterni, chiamati anche comandi estrinseci o programmi transienti, sono i programmi archiviati come file su disco. Prima di essere eseguiti questi programmi, devono essere caricati dal disco dentro la zona dei programmi transienti (il transient program area o TPA).

Esempi di questi comandi esterni sono FDISK, CHKDSK, BACKUP, FORMAT, MODE e SYS.

Quando un comando esterno ha terminato il suo lavoro viene scaricato dalla memoria, perciò, ogni volta che lo si vuole eseguire, deve essere ricaricato dal disco.

File batch

I file batch sono file di testo che contengono una lista di altri comandi interni, esterni o batch. Questi file sono elaborati da un interprete speciale che fa parte della parte transiente del COMMAND.COM.

L'interprete legge i file batch una riga alla volta e esegue tutte le operazioni specificate nell'ordine. Per interpretare un comando dell'utente, il COMMAND.COM cerca prima di capire a quale categoria appartiene il comando da eseguire.

Poi lo ricerca in ogni directory controllata, partendo dalla directory corrente del drive corrente e, successivamente nei drive e directory specificati dentro la stringa PATH dell'ambiente per un file con l'estensione .COM, poi con la estensione .EXE e infine per un estensione .BAT.

Funzione EXEC

Se il controllo fallisce per tutti e tre i tipi di file viene emesso un messaggio di errore. Se un file .COM o .EXE viene trovato, COMMAND.COM usa la funzione EXEC del DOS per caricarlo e eseguirlo. La funzione EXEC prepara una struttura dati speciale chiamata program segment prefix (PSP) sopra la porzione residente del COMMAND.COM nella zona dei programmi transienti (TPA).

PSP

Il PSP contiene i collegamenti e i vari puntatori necessari al progamma applicativo. Successivamente, la funzione EXEC carica il programma stesso subito sopra il PSP ed esegue le rilocazioni necessarie. Infine prepara i registri e trasferisce il controllo al punto di entrata del programma. Quando il programma transiente ha finito il suo lavoro, chiama una funzione del DOS per terminare il processo e libera la memoria, restituendo il controllo al programma che l'ha caricato (in questo caso, COMMAND.COM).

COME VIENE CARICATO IL DOS

Quando il sistema è inizializzato, l'esecuzione del programma, comincia all'indirizzo 0FFF0H che contiene un istruzione di jump machine per trasferire il controllo al system test code e alla procedura di ROM bootstrap.

Bootstrap

Il ROM bootstrap legge il programma di disk bootstrap dal primo settore del disco (settore di boot) dentro la memoria a un indirizzo arbitrario e trasferisce il controllo a questo indirizzo (il settore boot contiene anche una tabella di informazioni sul formato del disco).

Il programma di disk bootstrap va a vedere se il disco contiene una copia del DOS. Questo controllo viene fatto leggendo il primo settore della root directory per determinare se i primi due file sono IBMBIO.COM e IBMDOS.COM, nell'ordine. Se questi file non sono presenti, all'operatore viene chiesto di cambiare disco e premere qualsiasi tasto per riprovare.

File di sistema

Se i due file di sistema vengono trovati, il disk bootstrap li legge in memoria e trasferisce il controllo al punto iniziale di entrata di IBMBIO.COM (in qualche implementazione, il disk bootstrap legge solo l'IBMBIO.COM dentro la memoria, mentre l'IBMBIO.COM è responsabile per il caricamento del file IBMDOS.COM).

DOS DISCO INTERNO

Organizzazione dei dischi

I dischi DOS sono organizzati in accordo con un rigido schema che facilmente può essere capito e manipolato. Sebbene molti programmatori non avranno bisogno di accedere direttamente a delle aree particolari di un disco, conoscere la loro struttura interna porta ad una migliore comprensione dell'ambiente di lavoro.

Dal punto di vista applicativo il DOS presenta le unità disco come volumi logici che sono associati ad un drive code (A, B, C e così via), hanno un nome volume (opzionale), una root directory ed una o più sottodirectory e file. Inoltre, il DOS protegge il programmatore dalle caratteristiche fisiche del disco medio tramite l'INT 21H. Usando i suoi servizi, il programmatore può creare, aprire, scrivere, leggere, chiudere e cancellare i file in modo uniforme, senza curarsi del formato del drive del disco, della velocità, del numero di testine di lettura/scrittura, del numero di tracce e così via.

Tali richieste di operazioni su file in realtà passano attraverso due livelli di traduzione prima di giungere ad un trasferimento fisico di dati tra l'unità disco e l'accesso random della memoria.

Volumi logici

Ciascun volume logico DOS è diviso in diversi formati fissi, aree di controllo e aree di file. Il formato di ciascuna area di controllo può essere diverso a seconda del disk-drive e del costruttore considerato, ma tutte le informazioni occorrenti ad interpretare la struttura di un particolare disco possono essere trovate, nello stesso settore, sul disco stesso.

INTERRUZIONI DEL DOS

I due moduli DOS, IBMBIO.COM e IBMDOS.COM, facilitano l'uso del BIOS. Le routine di IBMBIO.COM forniscono un interfaccia di basso livello al BIOS.

IBMBIO.COM è un programma I/O handler che facilita la lettura dei dati dalle unità esterne dentro la memoria e la scrittura dei dati dalla memoria alle unità esterne.

File manager

IBMDOS.COM contiene un file manager e un numero di funzioni di servizio su come bloccare e sbloccare (blocking e de-blocking) i record sul disco. Quando un programma applicativo richiede INT 21H, l'operazione consegna l'informazione all'IBMDOS attraverso i contenuti dei registri. Per completare la richiesta IBMDOS traduce l'informazione in una o più chiamate all'IBMBIO.COM che, a sua volta, chiama il BIOS.

In totale ci sono nove servizi di interruzione, cinque dei quali sono veri interrupt poichè a ciascuno di essi è associato un compito definito; gli altri hanno compiti più generali:

INT 20H TERMINA IL PROGRAMMA

Questa interruzione termina l'esecuzione e restituisce il controllo al DOS. Poichè non chiude automaticamente i file, prima di uscire dal programma bisogna utilizzare la chiamata di funzione 10H del DOS INT 21H per chiudere tutti i file.

INT 21H RICHIESTA DI FUNZIONI DOS

Servizi di interruzione

Questa è l' interruzione principale del DOS, e richiede un codice di funzione dentro il registro AH.

INT 22H INDIRIZZO DI CHIUSURA

Specifica dove viene trasferito il controllo del computer quando il programma termina. Questo servizio viene normalmente usato per restituire il comando al COMMAND.COM

INT 23H INDIRIZZO DI GESTIONE DEL CTRL-BREAK

Pone termine al programma o al file di comandi batch in corso di esecuzione. Il tasto break a cui il DOS risponde con questa azione viene attivato premendo CTRL-BREAK sulle tastiere standard per PC o CTRL-C su tastiere generiche.

INT 24H VETTORE DI GESTIORE DEGLI ERRORI CRITICI

Punta alla routine di gestione dell'interruzione che viene richiamata ogni volta che viene scoperto un errore critico dal DOS. La routine ritorna al DOS dopo aver fatto ciò che noi abbiamo scelto di fare caricando in AL un valore che va da zero a due:

VALORE	DESCRIZIONE
0	Ignora l'errore, vai avanti
1	Ritenta l'operazione
2	Termina il programma

INT 25H E INT 26H LETTURA E SCRITTURA ASSOLUTA DEL DISCO.

Sono gli unici servizi del DOS che ignorano la struttura logica del disco. Si possono paragonare ai servizi per i dischi del ROM-BIOS, l'unica differenza consiste nel diverso metodo di numerazione con cui vengono individuati i settori. A seguito di un errore durante un'operazione di lettura o scrittura il carry flag (CF) è posto a 1 e nel registro AL viene caricato un valore la cui descrizione è riportata dalla precedente tabella.

INT 27H TERMINA MA RESTA IN LUOGO

Questo è uno dei servizi più importanti e interessanti del DOS. L'interruzione 27H termina un programma e anzichè cancellarlo dalla memoria lascia una parte specificata di esso in memoria e cambia l'informazione che riguarda la prima porzione di memoria disponibile,

CODICE ERRORE (ESA)	DESCRIZIONE
0	Tentata scrittura su disco protetto
1	Numero del lettore disco non valido
2	Lettore disco non pronto
4	Errore di parità
6	Errore Seek
7	Formato disco non riconosciuto
8	Settore non trovato
A	Errore di scrittura
B	Errore di lettura
C	Malfunzionamento generale

Codifica degli errori

perchè indichi il paragrafo che segue il programma residente.

La struttura di un programma residente di solito è composta delle seguenti parti:

Programma residente

1. Una sezione che ridefinisce le locazioni dentro la tabella dei servizi di interruzione.

2. Una procedura che viene praticata solo una volta e che esegue i seguenti passi:

- Sostituisce l'indirizzo nella tabella dei servizi di interruzione con il suo indirizzo. Stabilisce le dimensioni della porzione che deve rimanere residente.
- Usa una interruzione per dire al DOS di terminare questo programma e per collocare la porzione specificata nella memoria.

Una procedura che rimane residente può venire attivata, per esempio, dall'input dalla tastiera o in qualche caso, dall'orologio (timer clock).

La memoria adesso appare come segue:

Struttura della memoria

Vettori di interruzione
IBMBIO.COM e IBMDOS.COM
COMMAND.COM
Porzione residente del programma
Porzione di inizializzazione del programma (che sarà cancellata con il caricamento di un altro qualsiasi programma)
Il resto della memoria disponibile

Normalmente il programma resta residente finchè il DOS è residente. Ad ogni modo è preferibile l'uso dell'INT 21H funzione 31H perchè

vi permette di avere più memoria disponibile per il programma residente e di passare un codice di ritorno.

INT 2FH CONTROLLO SPOOL DI STAMPA

Questa interruzione costituisce ormai un metodo standard per le comunicazioni con qualsiasi spooler di stampa installato nel DOS. Sei funzioni numerate da zero a cinque compongono i servizi di controllo dello spooler di stampa che sono disponibili.

Chiamate di funzione

Tutte le chiamate di funzione del DOS sono richiamate dall'INT 21H. Segue una lista delle chiamate di funzione originarie del DOS 1.0 :

00 Termina il programma (uguale all' INT 20H)
01 Input dalla tastiera con eco
02 Visualizza l'output
03 Input seriale (comunicazione asincrona)
04 Output seriale (comunicazione asincrona)
05 Output su stampante
06 Tastiera e video diretti
07 Input diretto dalla tastiera senza eco
08 Input dalla tastiera senza eco
09 Visualizza stringa
0A Input dalla tastiera bufferizzato
0B Controllo dello status degli input dalla tastiera
0C Vuota il buffer della tastiera e invoca l'input
0D Reset del disco
0E Seleziona il disk-drive corrente
0F Apre un file
10 Chiude un file
11 Cerca il primo file sul disco
12 Cerca il prossimo file sul disco
13 Cancella un file
14 Legge record su file sequenziale
15 Scrive record su file sequenziale
16 Crea un file
17 Cambia nome a un file
19 Determina il disk drive attuale
1A Determina l'area di trasferimento sul disco
1B Prende informazioni sulla FAT del lettore corrente
1C Prende informazioni sulla FAT di qualsiasi lettore
21 Legge file ad accesso casuale
22 Scrive file ad accesso casuale
23 Determina la dimensione di un file
24 Fissa un campo di record su file ad accesso casuale
25 Fissa vettore delle interruzioni

26 Crea un segmento programma
27 Legge record su file ad accesso casuale
28 Scrive record su file ad accesso casuale
29 Analizza un nome di file
2A Acquisisce la data
(CX=Anno, DH=Mese, DL=Giorno, in binario)
2B Fissa la data
2C Acquisisce l'ora (CH=Ore, CL=Minuti, DH=Secondi, DL=Centesimi di secondo)
2D Fissa l'ora
2E Imposta la verifica scrittura disco

Le seguenti funzioni estese sono disponibili dalla versione DOS 2.0 e seguenti :

Chiamate di funzione per DOS 2.0

2F Acquisisce l'indirizzo del disk transfer area
(restuito attraverso ES:BX)
30 Acquisisce il numero della versione di DOS
(restuito attraverso AX)
31 Termina ma rimane residente (TSR)
33 Controllo del Ctrl/Break
35 Acquisisce il vettore delle interruzioni
36 Acquisisce lo spazio libero sul disco
38 Acquisisce informazioni del paese dipendente
39 Crea una directory (MKDIR)
3A Rimuove una directory (RMDIR)
3B Cambia la directory corrente
3C Crea un file
3D Apre un file
3E Chiude un file
3F Legge da un file
40 Scrive su un file
41 Cancella un file dalla directory
42 Sposta il puntatore di un file
43 Gestisce gli attributi di un file
44 Controllo dell' I/O per le unità
45 Duplica un file handle
46 Forza la duplicazione di un file handle
47 Acquisisce la directory corrente
48 Assegna la memoria
49 Libera la memoria assegnata
4A Modifica i blocchi di memoria assegnata
4B Carica/esegue un programma
4C Termina l'elaborazione (ritorna dalla chiamata di un programma)
4D Recupera il codice di ritorno di un sotto-processo
4E Inizia la ricerca di un file in una directory

4F Continua la ricerca di un file in un'altra directory
54 Acquisisce lo stato della verifica
56 Cambia nome a un file
57 Get/set la data e l'ora di un file

Le seguenti funzioni estese sono disponibili dalla versione DOS 3.0 e seguenti :

Chiamate di funzione per DOS 3.0

59 Acquisisce il codice di errore esteso
5A Crea un file temporaneo
5B Crea un file nuovo
5C Blocca/sblocca l'accesso ad un file
62 Acquisisce l'indirizzo del PSP

CAPITOLO 11

I FILE ESEGUIBILI COM E EXE

I programmi che "girano" in ambiente DOS rientrano in due formati base:

- I programmi COM che hanno una lunghezza massima di circa 64K.

- I programmi EXE che possono essere ampi quanto la disponibilità della memoria.

COM

I programmi COM sono più adatti per piccoli programmi, nei quali tutti i registri segmento contengono lo stesso valore, cioè il codice e i dati sono conglobati assieme.

EXE

I programmi EXE sono adatti per medi e grandi programmi, nei quali i registri segmento hanno valore diverso, cioè il codice, i dati e lo stack risiedono in segmenti separati.

Un programma tipo COM risiede sul disco come un'immagine di memoria assoluta, in un file con estensione .COM. Il file non ha una header o una qualsiasi informazione interna di identificazione.

Un programma EXE, invece, risiede sul disco in un tipo di file speciale con un unica header, una mappa di rilocazione, un checksum e ogni altra informazione che è (o può essere) usata dal DOS.

EXEC

Entrambi i programmi COM e EXE sono introdotti in memoria per l'esecuzione dallo stesso meccanismo: la funzione EXEC, che costituisce il caricatore del DOS. L'EXEC può essere chiamato con il filename di un programma caricato dal COMMAND.COM (il normale interprete di comando del DOS), da altri shell o interfacce utenti, o da un'altro programma precedentemente caricato dall'EXEC. Come affermato in precedenza, se c'è sufficiente memoria libera nell'area del programma transitorio, EXEC assegna un blocco di memoria per l'immagazzina-

PSP

mento del nuovo programma, costruisce un prefisso del segmento del programma (PSP) e la sua base e pone il programma nella memoria immediatamente sopra il PSP. Infine, EXEC sistema i registri segmento e lo stack e trasferisce il controllo al programma.

I programmi COM e EXE si riferiscono spesso a programmi transitori. Un programma transitorio che, mentre viene eseguit,o viene posto in un blocco di memoria, assume il controllo quasi totale delle risorse del sistema. Quando un programma termina, perchè ha completato il suo lavoro e sistematicamente esegue un ritorno al DOS, il blocco di memoria viene liberato e può essere usato per caricare il programma successivo.

IL PREFISSO DEL SEGMENTO DEL PROGRAMMA (PSP)

Il PSP è un'area riservata, di 256 byte, che viene sistemata dal DOS alla base del blocco di memoria destinato a un programma transitorio. Il PSP contiene collegamenti per il DOS che possono essere usati dal programma, informazioni salvate dal DOS per sue operazioni e delle informazioni che sono passate dal DOS ad un programma transitorio, per essere usate o no, come il programma stesso richiede.

Nel zona di memoria riservata ai programmi il PSP è collocato all'indirizzo di spiazzamento 0. Il programma è situato dopo il PSP all'indirizzo di spiazzamento 100H.

In un programma tipo COM tutti i registri di segmento puntano al PSP e il programma ha inizio all'indirizzo 100H.

Indirizzamento

Un programma tipo EXE, invece, inizia con i registri DS e ES che puntano al PSP e il programma ha inizio all'indirizzo posto in CS:IP in un punto normalmente diverso da 100H.

Benchè il DOS si sia sviluppato considerabilmente in questi ultimi anni, la struttura del PSP è ancora simile al suo antecedente CP/M.

Vediamo ora attraverso la seguente tabella che cosa contiene il PSP:

Offset	Contenuto
0000H	L'istruzione INT 20H.
0002H	Dimensione in paragrafi della memoria.
0004H	Byte riservato DOS.

Contenuto del PSP

Offset	Contenuto
0005H	Chiamata FAR al gestore di funzioni DOS.
000AH	Vettore interruzione di terminazione (INT 22H).
000EH	Vettore interruzione CTRL-C e CTRL-BREAK.
0012H	Vettore interruzione per errori critici.
0016H	Zona riservata DOS.
002CH	Segmento del blocco di ambiente.
002EH	Area di lavoro del DOS.
0050H	Istruzione INT 21H e RETF.
0053H	Zona riservata DOS.
0055H	Estensione File Control Block (FCB) 1.
005CH	FCB 1.
0065H	Estensione FCB 2.
006CH	FCB 2.
0080H	Area di trasferimento del disco (DTA).
00FFH	Fine.

Il primo byte della DTA contiene il numero dei caratteri che seguono, sulla linea di comando, il nome del programma (ma non compreso il CARRIAGE RETURN), seguono i caratteri, poi il CARRIAGE RETURN.

Se qualche nome di file era stato battuto sulla linea di comando, il primo nome sarà inserito nel FCB 1 e il secondo nome sarà inserito nel FCB 2.

Stringhe ASCIIZ

Il blocco di ambiente contiene una serie di stringhe ASCIIZ (questo significa che tutte le stringhe sono terminate con un byte 0), con un byte addizionale 0 alla fine dell'ultima stringa. Nel DOS 3.0 e nelle versioni seguenti, questo byte 0 e seguito da 2 byte in più, seguiti dallo intero pathname ASCIIZ usato per caricare il programma.

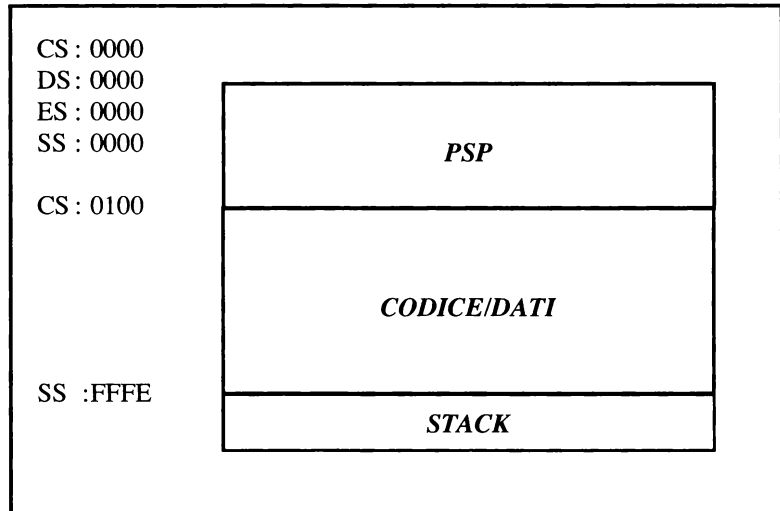
INTRODUZIONE AI PROGRAMMI COM

Programmi COM

I programmi di estensione COM sono file eseguibili che contengono un immagine assoluta del programma da eseguire, senza informazioni per la rilocazione. Essi sono più compatti e quindi sono caricati più velocemente dei file EXE.

I programmi COM sono caricati immediatamente dopo il PSP con spiazzamento nel segmento di 0100H, che è la lunghezza del PSP. Questo indirizzo deve contenere un'istruzione eseguibile. La massima lun-

ghezza di un programma COM è di 65536 byte, a cui vanno tolti i 256 byte del PSP e uno stack minimo di 2 byte.

Quando il controllo è trasferito dal DOS al programma COM, tutti i registri di segmento puntano al PSP. Il puntatore stack (SP) contiene un indirizzo di 2 byte minore rispetto all'indirizzo più alto possibile nel segmento (normalmente FFFEH).

Quando un programma COM ha finito la propria esecuzione, può rimettere il controllo al DOS tramite: la funzione 4CH dell' interrupt 21H, la funzione 00H dello stesso interrupt, un interrupt 20H, o un RETURN NEAR.

Se un programma COM è creato collegando con LINK alcuni moduli oggetto, tutti questi moduli devono avere lo stesso nome del segmento codice, lo stesso nome classe e il primo modulo deve contenere il punto di entrata nel programma.

Tutte le procedure nel programma COM devono essere procedure NEAR, poichè tutti i programmi risiedono in un segmento solo.

UN ESEMPIO DI PROGRAMMA COM

Il programma DUMMYCOM esposto nella figura seguente ci mostra la struttura di un semplice programma in linguaggio assembler che è destinato a divenire un file COM. Questo programma è semplice e breve,

in quanto una alta proporzione del codice sorgente è in realtà contenuta nelle direttive MASM che non risultano in nessun codice eseguibile.

La direttiva NAME fornisce semplicemente il nome del modulo da usare durante il processo di LINK. Se il comando NAME non è presente nel file sorgente, i primi sei caratteri del testo forniti nel TITLE sono usati come nome del modulo. Se nessuno di questi è presente, il LINK userà il filename.

LINK

La direttiva PAGE definisce la lunghezza e la larghezza della pagina a 60 linee e a 132 colonne rispettivamente. Se la direttiva PAGE è usata senza operandi, indica una nuova pagina.

Programma DUMMYCOM

Direttive

La direttiva TITLE specifica una stringa di caratteri che viene posta al di sopra di ciascuna pagina.

NAME

Dopo pochi commenti è esposta la direttiva EQU che definisce un codice per documentare meglio il programma.

Poi la direttiva SEGMENT che inizia un segmento con il nome cseg e gli attributi PARA, PUBLIC e 'CODE'.

PAGE

La prossima riga contiene la direttiva ASSUME. Questa informa il MASM su quali sono i contenuti dei registri di segmento per permettergli di ottimizzare il programma dove possibile. Nota che l'ASSUME esposto non si prende cura di caricare i registri di segmento con i propri valori. Esso notifica al MASM solo dell'intenzione del programmatore.

TITLE

EQU

Poichè questo programma deve essere convertito in un file COM, tutto il suo codice eseguibile e i dati devono essere posti nel segmento codice. Il programma deve anche avere una direttiva ORG 0100H. Nei casi in cui i dati sono presenti è usuale porli subito dopo la prima istruzione eseguita, cioè un JMP.

SEGMENT

ASSUME

Dopo i dati troviamo la procedura start che inizia con la direttiva PROC NEAR. Il NEAR significa che la procedura può essere chiamata solamente da altre procedure nello stesso segmento. Nei programmi COM, tutte le procedure sono di tipo NEAR.

Per questo esempio la procedura è molto semplice. Essa chiama solo la funzione 4CH del DOS per terminare il programma.

ORG 0100 H

La fine della procedura start è segnata dalla direttiva ENDP, la fine del segmento cseg con una ENDS, e la fine del programma con END che deve avere una label per definire il punto di entrata nel programma.

```
                              DUMMYCOM. ASM

Esempio di programma assembler per creare un file COM

                title     DUMMYCOM.ASM
                page      60,132                  ;60 righe e 132 colonne

TERMINA         equ       4Ch                     ;termina programma

                              CODE

cseg            segment para public "CODE"
                assume cs:cseg,ds:cseg,es:cseg,ss:cseg

                org 0100h                         ;riserva lo spazio per il PSP

; Dopo il caricamento di un file COM,
; cs:ip punta a cs:entry,
; ss:sp punta a cseg:FFFEh
; ds e es puntano to cseg:0000 (il PSP).

entry:          jmp       start                   ;salta la zona dati

                              DATA

; metti qui i dati

                              CODE

start           proc      near

; metti qui il codice

exit:           mov       al,00h                  ;exit code
                mov       ah,TERMINA              ;termina programma
                int       21h                     ;chiamata funzione DOS

start           endp

cseg            ends
                end       entry                   ;definisce l'entry point
                                                  ;del programma
```

Figura 11.1 - Esempio di programma assembler per creare un file COM.

INTRODUZIONE AI PROGRAMMI EXE

PROC NEAR

ENDP

Programmi EXE

Abbiamo esaminato un programma tipo COM. Trattiamo ora i programmi EXE. I programmi COM hanno degli svantaggi definiti e, come risultato, i programmi più impegnativi sono sviluppati come programmi EXE.

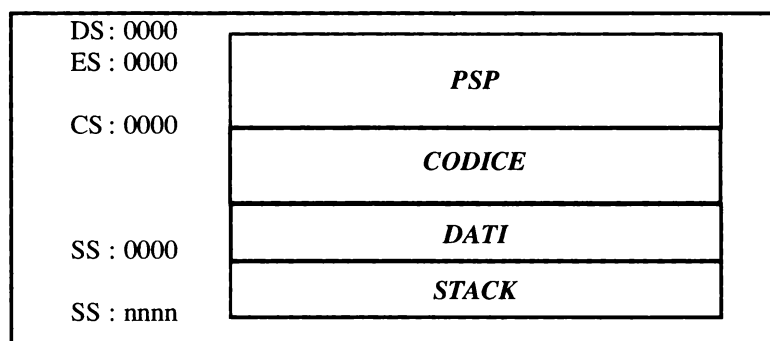
Mentre i programmi COM sono limitati a una lunghezza massima di 64K e a un solo segmento, i programmi EXE possono essere praticamente illimitati. I programmi EXE collocano i codici, i dati e lo stack in segmenti separati.

Un programma EXE è spesso inserito nella memoria caricandolo immediatamente sopra il PSP, sebbene possono essere vari gli ordini dei codici, dei dati e dello stack.

Prima che il DOS trasferisca il controllo al programma, i valori iniziali del registro segmento codice CS e del puntatore di istruzione IP vengono calcolati dall'informazione presente nell'header del file EXE e dall'indirizzo del programma caricato. Queste informazioni giungono da un direttiva END nel codice sorgente di uno dei moduli del programma.

I registri segmenti dati DS e extra ES puntano al PSP, così il programma può accedere all'environment block, alla linea di comando e alle altre informazioni utili.

I contenuti del registro segmento di stack SS e dello stack pointer SP derivano dall'header. Questa informazione giunge dalla dichiarazione di un segmento con l'attributo STACK nel codice sorgente del programma. L'input al LINK per un programma di tipo EXE può essere composto da moduli oggetto separati. Ciascun modulo può usare un unico nome di segmento codice e le procedure possono essere o NEAR o FAR, a seconda della sorgente. E' obbligatorio avere un solo punto di entrata definito con la direttiva END.

Indirizzo	Segmento
DS : 0000 ES : 0000	*PSP*
CS : 0000	*CODICE*
	DATI
SS : 0000	*STACK*
SS : nnnn	

I registri

Il programma DUMMYEXE ci mostra la fondamentale struttura di un programma del tipo EXE. Come minimo esso deve avere un nome del modulo, un segmento codice, un segmento stack e una procedura primaria che riceva il controllo da DOS dopo il caricamento del programma. Il programma DUMMYEXE contiene anche un segmento dati.

Le direttive NAME, TITLE e PAGE sono già state descritte e vengono usate nello stesso modo visto in DUMMYCOM.

Dopo pochi commenti e dopo gli esposti EQU, giungiamo ad un segmento dati chiamato dseg che inizia con la direttiva SEGMENT e finisce con ENDS. Essa normalmente contiene i dati usati dal programma, assenti in questo caso.

Programma DUMMYEXE

In seguito si ha una dichiarazione di un segmento codice che inizia con la direttiva SEGMENT seguita da una direttiva ASSUME. Nota che, a differenza dell'equivalente esposto nel DUMMYCOM, l'ASSUME specifica i nomi dei vari e differenti segmenti.

Direttive

Nel segmento codice, la procedura start è dichiarata dal comando PROC. Abbiamo dato l'attributo NEAR come esempio, ma nei programmi EXE l'attributo può essere anche FAR.

SEGMENT
ENDS

All'interno della procedura, inizializziamo per primi i registri DS e ES alla base della zona dati, come definito dal MASM della direttiva ASSUME. Notate che con questa azione abbiamo perso l'indirizzo del PSP, che era in DS.

ASSUME

Anche questa procedura di esempio è molto semplice. Dopo l'inizializzazione di DS e ES essa chiama solo la funzione 4CH del DOS per terminare il programma.

PROC
NEAR
FAR

La fine della procedura start è segnata dalla direttiva ENDP e la fine del segmento cseg con una ENDS.

Poi il segmento stack chiamato sseg che inizia con la direttiva SEGMENT e finisce con ENDS. Prima che il DOS trasferisca il controllo ad un programma EXE, sistema i registri SP e SS in accordo con l'informazione contenuta nell'header del file EXE e la locazione del programma in memoria.

L'END termina il programma, dicendo al MASM che si trova alla fine del file sorgente e provvede all'etichettatura del punto di entrata del programma da DOS.

```
                         DUMMYEXE.ASM

Esempio di programma assembler per creare un file EXE

          title   DUMMIEXE.ASM
          page    60,132            ;60 righe e 132 colonne

TERMINA   equ     4Ch               ;termina programma

                            DATA

dseg      segment para public "DATA"

; metti qui i dati

dseg      ends

                            CODE

cseg      segment para public "CODE"
          assume cs:cseg,ds:dseg,es:dseg,ss:sseg

; Dopo il caricamento di un file EXE,
; cs:ip punta a start,
; ss:sp punta a stktop,
; ds e es puntano al PSP.

start     proc    near

          mov     ax,dseg           ;punta al nostro
          mov     ds,ax             ;segmento dati
          mov     es,ax             ;

; metti qui il codice

exit:     mov     al,00h            ;exit code
          mov     ah,TERMINA        ;termina programma
          int     21h               ;chiamata funzione DOS

start     endp

cseg      ends

                            STACK

sseg      segment para stack 'STACK'

stkend    dw      27 dup (?)        ;lo stack termina qui
stktop    dw      ?                 ;lo stack comincia qui

sseg      ends
          end     start             ;definisce l'entry point
                                    ;del programma
```

Figura 11.2 - Esempio di programma assembler per creare un file EXE

CAPITOLO 12

MEZZI DI SVILUPPO SOFTWARE

Per creare un programma funzionante sul PC bisogna usare le utility MICROSOFT:

Utility MICROSOFT

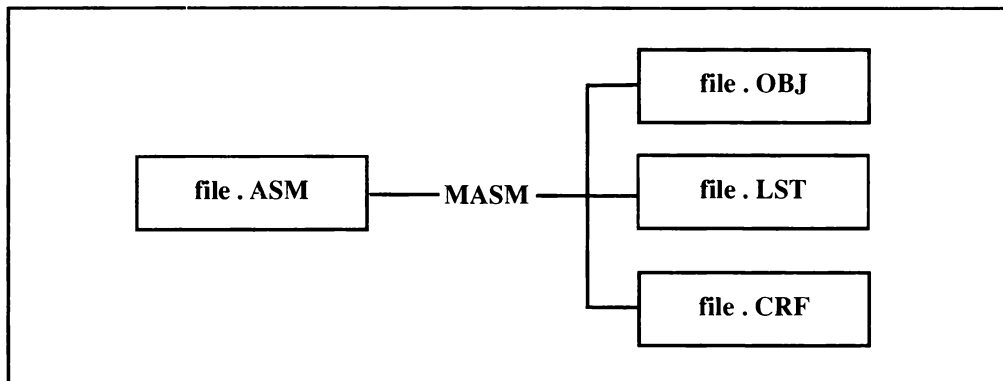
1. Scrivere il codice sorgente con qualche editore di testo.
2. Convertire il sorgente, tramite MASM, in file oggetto.
3. In opzione, creare un file di cross-reference con CREF.
4. Generare le librerie di moduli oggetti con LIB.
5. Usare LINK per collegare i file oggetti e le librerie.
6. Per programmi piccoli, convertire il file EXE in file COM.
7. Se il programma non funziona, usare un debugger come DEBUG, SYMDEB o CODEVIEW per trovare gli errori.

Nota: In questo capitolo il simbolo ↵ significa il tasto d'invio sulla tastiera PC.

IL MACRO-ASSEMBLATORE: MASM

file OBJ

MASM è un programma che converte il codice sorgente in istruzioni del linguaggio macchina, tuttavia queste istruzioni non vengono fornite in una forma eseguibile, ma in una forma detta codice oggetto. Questi file OBJ devono essere combinati con altri file OBJ e convertiti in un programma caricabile.

Tale lavoro viene descritto nella sezione LINK.

CARICANDO MASM

L'assemblatore può essere attivato in due modi:

1. Con i parametri forniti interattivamente come risposta alle domande. Ad esempio:

   Al prompt DOS, digita MASM. Il programma chiederà:

   Source filename [.ASM]: digita NOME ↵
   Object filename [NOME.OBJ]: digita NOME ↵
   Source listing [NUL.LST]: digita NOME ↵
   Cross-reference [NUL.CRF]: digita NOME ↵

   Tra i caratteri [ ] vengono scritti i parametri di default.

Esempio MASM

Nell'esempio MASM converte il file NOME.ASM e genera il file oggetto NOME.OBJ, il file lista NOME.LST e un file di cross-reference NOME.CRF.

2. Con i parametri specificati nella linea di comando nel formato:

MASM /sw1 /sw2 sorgente,oggetto,lista,crossref; ↵

Nota il punto e virgola: questo obbliga MASM a terminare la linea di comando ed usare i default per i parametri non specificati.
Quindi è possibile digitare solo:

MASM NOME; ↵

con lo stesso effetto di questa linea:

MASM NOME.ASM,NOME.OBJ,NUL.LST,NUL.CRF; ↵

Per avere lo stesso risultato dell'esempio interattivo bisogna digitare:

MASM NOME.ASM,NOME.OBJ,NOME.LST,NOME.CRF; ↵

Esempio MASM

Gli switch sono rappresentati da uno o più caratteri preceduti da una barra e sono usati per specificare i formati delle liste, delle tavole di simboli, e la generazione di segmenti e codici per il coprocessore numerico.

Per chiedere informazioni sulle funzioni degli switch, digitare:

MASM /HE ↵

Sullo schermo apparirà una descrizione della linea di comando MASM, seguito da tutte le opzioni:

Usage

```
Usage: MASM /options source(.asm),
                    [out(.obj)],[list(.lst)],[cref(.crf)][;]

/a                   Alphabetize segments
/b<number>           Set I/O buffer size, 1-63 (in 1K blocks)
/c                   Generate cross-reference
/d                   Generate pass 1 listing
/D<sym>[=<val>]      Define symbol
/e                   Emulate floating point instr. and IEEE format
/I<path>             Search directory for include files
/l[a]                Generate listing, a-list all
/M{lxu}              Pres. case of labels: l-All, x-Glob, u-Upcs. Glob.
/n                   Suppress symbol tables in listing
/p                   Check for pure code
/s                   Order segments sequentially
/t                   Suppress messages for successful assembly
/v                   Display extra source statistics
/w{012}              Set warning level: 0-None, 1-Serious, 2-Advisory
/X                   List false conditionals
/z                   Display source line for error messages
/Zi                  Generate symbolic information for CodeView
/Zd                  Generate line-number information
```

IL FORMATO DEL FILE SORGENTE

In generale, le righe sorgenti presentano il seguente formato:

nome[:] mnemonico operando, operando ; commento

Formato delle righe sorgenti

dove:

nome	una label, una variabile o un simbolo.
mnemonico	un'istruzione 8088 o una direttiva MASM.
operando	una espressione.
commento	una descrizione dell'operazione, prefisso con un punto e virgola.

Ad esempio:

```
inizio: mov cx,ss:[0011h] ; carica la lunghezza in cx
```

Possono anche esserci delle righe con direttive MASM.
Le righe contenenti le direttive sono di cinque tipi:

Direttive per organizzare il programma.

COMMENT	NAME
EQU	PUBLIC
EXTRN	.RADIX
INCLUDE	

Direttive

Direttive per organizzare la memoria.

ASSUME	LABEL
DB DW DD DQ DT	ORG
END	PROC
ENDP	RECORD
ENDS	SEGMENT
EVEN	STRUC
GROUP	

Direttive per definire blocchi di programma per l'assemblaggio condizionale.

IF IFE IF1 IF2 IFDEF IFNDEF
IFB IFNB IFIDN IFDIF
ELSE
ENDIF

Direttive per creare blocchi di programmi MACRO.

ENDM	MACRO
EXITM	PURGE
IRP IRPC	REPT
LOCAL	
operatori macro:	& ; ; ! %

Direttive

Direttive per definire il formato e il contenuto del file lista.

	CREF	SFCOND
	LALL	SUBTTL
	LFCOND	TFCOND
	LIST	TITLE
	%OUT	XALL
PAGE	.XCREF	
SALL	XLIST	

IL FORMATO DEL FILE LISTA

Nel file lista le righe hanno lo stesso formato del file sorgente, preceduto da alcuni dati in più:

00A4 36: 8B 0E 0011 R C ...riga del file sorgente...

Dati aggiuntivi

00A4	rappresenta un indirizzo in memoria.
36: 8B 0E 0011	sono i codici generati (mov cx,ss:[0011h]).
36:	è il prefisso di segmento stack.
R	indica un valore da risolvere con LINK.
C	un eventuale carattere in questa posizione indica file sorgenti prodotti: C con la direttiva INCLUDE. + dopo l'espansione di una MACRO.

Alla fine del file lista ci sarà una tavola di simboli e informazioni utili, come nell'esempio della pagina seguente:

Macros:

N a m e	Lines
NOMEMAC	1

Structures and Records:

Strutture

Name	Width Shift	# fields Width	Mask	Initial
NOMEREC	0008	0003		
FIELD1	0006	0002	00C0	0040
FIELD2	0002	0004	003C	0028
FIELD3	0000	0002	0003	0000
NOMESTRU	0006	0003		
MAX	0000			
MIN	0001			
INIZ	0002			

Segments and Groups:

Name	Length	Align	Combine	Class
CODESEG	0139	PARA	PUBLIC	'CODE'

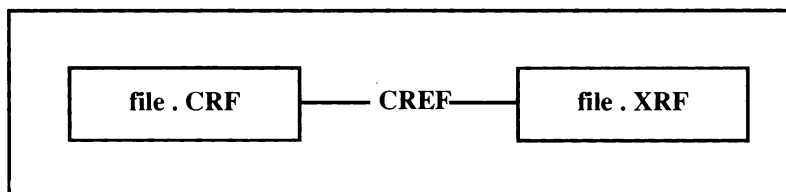
Symbols:

Simboli e informazioni

Name	Type	Value	Attr
BUFFER	L BYTE	0103	CSEG
INIZIA	L NEAR	0100	CSEG
EOF	NUMBER	001A	
FIELD1		0006	
FIELD2		0002	
FIELD3		0000	
REC1	L BYTE	010A	CSEG
STRUC1	L FWORD	0104	CSEG
USCITA	L NEAR	0135	CSEG
@FILENAME	TEXT NOME		

65 Source Lines
65 Total Lines
9 Symbols
50530 + 396782 Bytes symbol space free
0 Warning Errors
0 Severe Errors

IL GENERATORE DI FILE CROSS-REFERENCE: CREF

CREF è un programma che converte il file CRF prodotto da MASM in un file cross-reference REF.

Caricando cref

CREF può essere attivato in due modi:

1. Con i parametri forniti interattivamente come risposta alle domande. Ad esempio:

Al prompt DOS, digita CREF. Il programma chiederà:

CREF

Cross-reference [.CRF]:	digita NOME ↵
Listing [NOME.REF]:	digita ↵

2. Con i parametri specificati nella linea di comando secondo il seguente formato:

CREF file.CRF file.REF; ↵

Come MASM, il punto e virgola obbliga CREF a terminare la linea di comando e a usare i default per i parametri non specificati. Quindi è possibile digitare solo:

Esempio

CREF NOME; ↵

con lo stesso effetto di questa linea:

CREF NOME.CRF NOME.REF ↵

Il formato del file ref

Il file REF contiene:

1. Una lista ordinata alfabeticamente di tutti i simboli del file ASM con il numero di linea sorgente in cui sono generati.
2. Una cross-reference dei simboli che riporta i numeri delle linee sorgenti in cui il simbolo è definito, usato e modificato. Ad esempio:

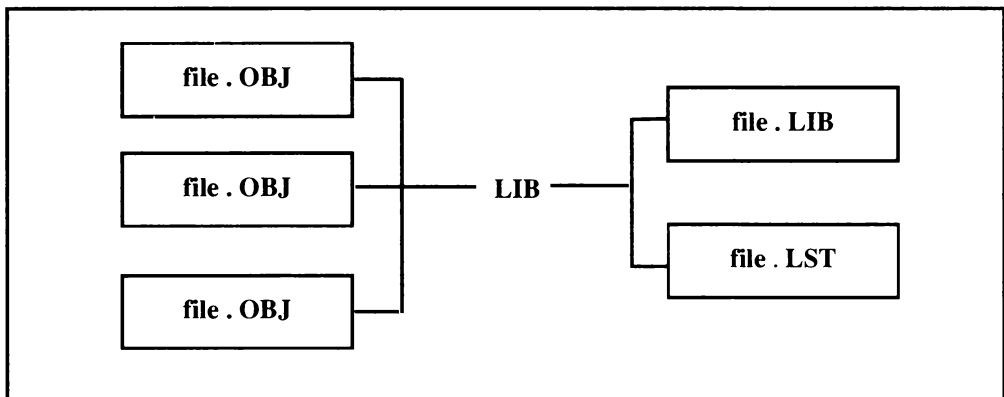
Formato del file

PROVA .... 134#   237   241   544+

Il simbolo PROVA è: definito nella riga 134,
usato nelle righe 237 e 241,
e modificato nella riga 544.

IL GENERATORE DELLE LIBRERIE: LIB

LIB e LINK

Risulta un pò scomodo avere tanti file oggetto sul disco e specificare quali di essi debbano essere combinati nel programma. La migliore soluzione è quella di creare una libreria di oggetti riuniti insieme in un unico file LIB. In seguito questa libreria viene perlustrata dal LINK per estrarne automaticamente gli oggetti necessari a completare un programma.

LIB può essere impiegato per tre diversi scopi:

1. Per esaminare i contenuti di una libreria già esistente.

2. Per sostituire dei moduli individuali nella libreria quando è necessario modificare un oggetto.
3. Per creare una nuova libreria.

LIB può essere attivato in due modi:

Utilizzo del file LIB

1. Con i parametri forniti interattivamente come risposta alle domande. Ad esempio:

Al prompt DOS, digita LIB. Il programma chiederà:

Library name: Digita NOME ↵

A questo punto, se NOME.LIB non esiste, il programma chiederà:

Library does not exist. Create ?

Digita N per ritornare al DOS,
oppure Y per creare la libreria.

Il programma risponderà:

Operations ?

A questo punto abbiamo la possibilità di usare i comandi:

Sequenza del file LIB

+ per appendere un file OBJ oppure LIB alla libreria.
- per rimuovere un modulo oggetto dalla libreria.
* per copiare un modulo e salvarlo nel file OBJ.

2. Con i parametri specificati nella linea di comando. Il formato è:

LIB nomelib.LIB comandi,filelist

Per esaminare i contenuti di una libreria:

LIB nomelib.LIB,CON;

Per stampare i contenuti di una libreria:

LIB nomelib.LIB,PRN;

Opzioni di LIB

Per rimuovere un modulo ABC dalla libreria:

LIB nomelib.LIB-ABC;

Per appendere un modulo ABC.OBJ alla libreria:

LIB nomelib.LIB+ABC;

Per sostituire un modulo ABC nella libreria:

LIB nomelib.LIB-ABC+ABC;

Per copiare un modulo ABC della libreria e salvarlo nel file ABC.OBJ:

LIB nomelib.LIB*ABC;

Per rimuovere un modulo ABC dalla libreria e salvarlo nel file ABC.OBJ:

LIB nomelib.LIB*ABC-ABC;

I contenuti della libreria

La libreria è composta da:

Contenuti

1. Moduli oggetto inseriti con l'operazione append.
2. Un indice dei moduli per l'uso di LIB.

Il formato della visualizzazione dei contenuti

Per esaminare i contenuti di una libreria, digita:

LIB nomelib.LIB,CON;

Ogni modulo oggetto copiato da un file OBJ può contenere tante label pubbliche, quindi la lista dei contenuti della libreria consta di due parti:

Formato libreria

1. Una lista alfabetica di tutti i simboli pubblici, ognuno associato al rispettivo modulo oggetto.
2. Una lista alfabetica di tutti i moduli oggetto, il loro offset nella libreria e la lunghezza in byte dei moduli. Ogni nome modulo è seguito da un sommario dei nomi pubblici dichiarati nel modulo.

Un esempio:

nome1 .....modulo1 nome3 .... modulo1
nome2 .....modulo1 nome4 .... modulo1
exit ....modulo2 nome5 .... modulo1

Esempio

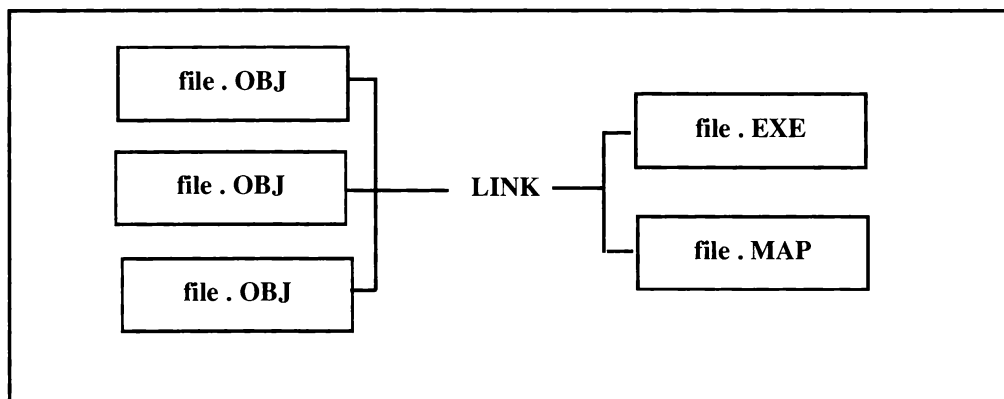
modulo1 Offset: 00000010H Code and data size: BDEH
nome1 nome2 nome3 nome4
nome5

modulo2 Offset: 000010F0H Code and data size: 1850H
exit

IL LINKER DEI FILE OBJ: LINK

LINK

LINK concatena file oggetti generati da MASM o da altri compilatori di linguaggi ad alto livello, risolve i riferimenti esterni e produce un file rilocabile tipo EXE e un file lista tipo MAP.

Il file rilocabile EXE può essere caricato ed eseguito all'indirizzo di memoria scelto dal sistema operativo DOS.

Caricando link

Il linker può essere attivato in tre modi:

1. Con i parametri forniti interattivamente come risposta alle domande. Ad esempio:

Al prompt DOS, digita LINK. Il programma chiederà:

Attivazione del file LINK

Object Modules [.OBJ]:	digita NOME1,NOME2
Run File [NOME1.EXE]:	digita ↵
List File [NUL.MAP]:	digita NOME1 ↵
Libraries [.LIB]:	digita ↵

Tra i caratteri [ ] vengono scritti i parametri di default.

Nell'esempio, LINK concatena i file NOME1.OBJ e NOME2.OBJ e genera i file NOME1.EXE e NOME1.MAP. Non è usata nessuna libreria.

2. Con i parametri specificati nella linea di comando nel formato:

```
LINK        oggetto1+oggetto2,fileexe,filemap,
            filelib1+filelib2 /sw1 /sw2 ; ↵
```

3. Con i parametri specificati in un file di risposte automatiche. La sintassi del comando usato è:

Sintassi

```
LINK @filerisp ↵
```

Nota che il nome del file va preceduto dal simbolo @.

Gli switch sono rappresentati da uno o più caratteri preceduti da una barra e sono usati per specificare la generazione dei file EXE e MAP.

Per chiedere informazioni sulle funzioni degli switch, digitare:

```
LINK /HE ↵
```

Sullo schermo verrà visualizzata una descrizione delle opzioni valide:

Visualizzazioni

/BATCH	/CODEVIEW
/CPARMAXALLOC	/DOSSEG
/DSALLOCATE	/EXEPACK
/FARCALLTRANSLATION	/HELP
/HIGH	/INFORMATION
/LINENUMBERS	/MAP
/NODEFAULTLIBRARYSEARCH	/NOFARCALLTRANS.
/NOGROUPASSOCIATION	/NOIGNORECASE
/NOPACKCODE	/OVERLAYINTERRUPT
/PACKCODE	/PAUSE
/QUICKLIBRARY	/SEGMENTS
/STACK	

Il link dei moduli per file tipo com

Link dei .COM

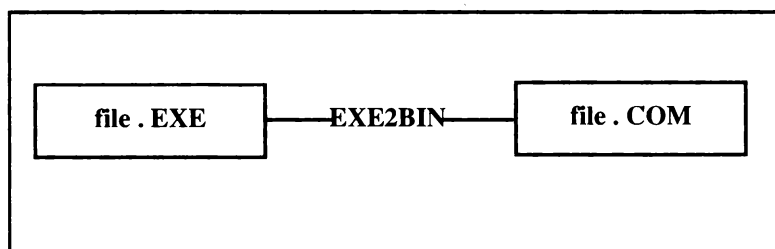
Quando LINK genera un file EXE che sarà convertito con EXE2BIN in file COM, il seguente messaggio verrà visualizzato:

Warning: no stack segment

Questo messagio può essere ignorato.

Il convertitore dei file exe in file com: exe2bin

File .EXE

EXE2BIN converte file da formato esadecimale EXE, in formato binario BIN. I file EXE, prodotti da LINK, devono avere una massima lunghezza di codice e dati combinati di 64 kilobyte ed essere senza lo STACK.

Caricando exe2bin

EXE2BIN file1.EXE,file2.ext ↵

EXE2BIN

L'estensione EXE può essere omessa, in quanto viene assunta per default. Si può anche omettere l'estensione del file2, in questo caso viene assunta automaticamente l'estensione BIN, ma normalmente si utilizza EXE2BIN per generare file di tipo COM. Il prossimo esempio mostra come convertire il file NOME.EXE in file NOME.COM:

EXE2BIN NOME,NOME.COM ↵

E' possibile omettere il nome file2.ext. Per convertire il file NOME.EXE in file NOME.BIN basta digitare:

EXE2BIN NOME ↵

I DEBUGGER: DEBUG, SYMDEB, CODEVIEW

DEBUG

Quasi tutte le versioni di DOS includono il DEBUG, un debugger adeguato ai piccoli progetti, perciò in questo libro ci soffermeremo solo per un momento sul soggetto dei debugger MICROSOFT più avanzati.

SYMDEB

Per svilupare programmi più complicati il debugger simbolico SYMDEB è incluso nelle utility del kit MASM. I vantaggi del SYMDEB sono:

Debugger simbolico

1. SYMDEB può leggere un file SYM (bisogna usare la utility MAPSYM per convertire un file MAP creato dal LINK in file SYM) per visualizzare i simboli pubblici nel corso dell'operazione di debug.

2. Permette anche la visualizzazione delle righe sorgenti per debuggare programmi scritti in linguaggi ad alto livello.

3. Ha la possibilità di avere schermi separati per le uscite del SYMDEB e del programma in debug.

4. Ha tutti i comandi del DEBUG, quindi è facile apprenderne l'utilizzo. Dispone inoltre di ulteriori comandi.

5. Fornisce informazioni sui comandi SYMDEB quando il tasto punto interrogativo viene premuto.

CODEVIEW

CODEVIEW, incluso nei più recenti kit dei linguaggi MICROSOFT, è molto più avanzato. Tra le facilitazioni vi sono:

Utility CODEVIEW

1. Visualizzazione tramite Window, anche a colori.

2. Selezione dei comandi da Menù o dalla linea di comando.

3. Operazione con o senza Mouse e con tasti di funzione.

4. Aiuto fornito dal sistema Help sempre disponibile.

5. Debug simbolico e delle righe sorgenti.

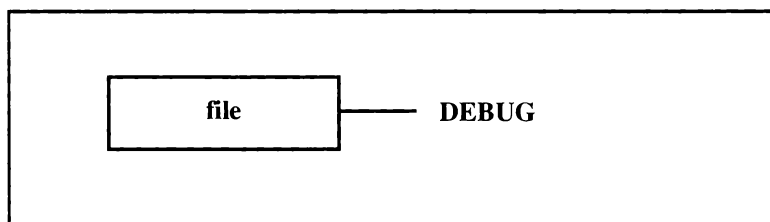
6. Comandi contenenti espressioni complicate.

7. I Tracepoint per eseguire un programma fino al cambiamento di un valore in memoria.

Debugging

8. I Watchpoint per eseguire un programma fino a quando la condizione di una espressione è vera.

9. La possibilità di rieseguire un programma senza la necessità di ricaricarlo.

Debug

Caricando debug

Il formato della riga di comando per il DEBUG è:

DEBUG nomefile,arg1,arg2,... ↵

Per attivare il DEBUG senza caricare un file:

Attivazione del DEBUG

DEBUG ↵

Per attivare il DEBUG e caricare il file NOME.EXE in memoria:

DEBUG NOME.EXE ↵

Per attivare il DEBUG, caricare il file NOME.EXE in memoria e fornire due nomi come parametri al programma caricato in memoria:

DEBUG NOME.EXE, NOME1, NOME2 ↵

I parametri costituiscono il resto della linea di comando che viene specificata quando il file NOME.EXE è eseguito normalmente, senza DEBUG.

Lo schermo iniziale del debug

La videata iniziale presentata dal DEBUG non è di molto aiuto. Comprende solo il carattere meno. Ma è sufficente digitare R per disporre della seguente immagine:

```
-R
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1A9A ES=1A9A SS=1A9A CS=1A9A IP=0100 NV UP EI PL NZ NA PO NC

1A9A:0100   A1BE0B        MOV     AX,[0BBE]          DS:0BBE=26F4
```

Pagina dello schermo

L'immagine riprodotta visualizza i contenuti di tutti i registri, i flag di status, la prossima istruzione da eseguire (in codice macchina e mnemonico) e l'indirizzo a cui è locata.

Altri comandi sono mostrati in basso.

I comandi di debug

Comandi di debug

A	Assemble	Assembla mnemonici in memoria.
C	Compare	Confronta due aree di memoria.
D	Dump	Lista un'area di memoria.
E	Enter	Pone dati in memoria.
F	Fill	Pone dati ripetuti in memoria.
G	Go	Esegue il programma in memoria.
H	Hex	Calcola N+M e N-M in esadecimale.
I	Input	Legge un byte da una porta I/O.
L	Load	Carica dati dal disco alla memoria.
M	Move	Sposta un'area di memoria.
N	Name	Nomina il file per i comandi L e W.
O	Output	Scrive un byte in una porta I/O.
P	Proceed	Prosegue dopo una CALL o INT.
Q	Quit	Esce dal DEBUG.
R	Register	Visualizza/cambia un registro.

S	Search	Ricerca una lista di byte.
T	Trace	Esegue una o più istruzioni.
U	Unassemble	Disassembla la memoria in istruzioni.
W	Write	Scrive dati dalla memoria al disco.

CAPITOLO 13

UN PROGRAMMA DI DIMOSTRAZIONE

DESCRIZIONE DEL PROGRAMMA

Il file batch MKMAIN.BAT elabora il file MAIN.ASM per produrre il programma di dimonstrazioneMAIN.EXE.

Questo programma:

- Installa una routine di servizio interrupt.
- Pulisce lo schermo.
- Visualizza una cornice.
- Suona toni con frequenze diverse.
- Posiziona il cursore.
- Legge dalla ROM del BIOS.
- Visualizza la data del BIOS.
- Attende che un tasto della tastiera sia premuto.
- Posiziona il cursore.
- Ritorna al DOS.

DESCRIZIONE DELLE PROCEDURE DEL PROGRAMMA

CLRSCRN Usa l'interrupt BIOS 10H per pulire lo schermo.

LINEBEG Scrive direttamente nella memoria video.

LINEINT Scrive direttamente nella memoria video.

CORNICE Usa l'interrupt BIOS 10H per leggere il modo del video.

SUONOV	Scrive direttamente sul PIT e PPI per controllare l'altoparlante.
SUONO	Chiama SUONOV per suonare toni con frequenze diverse.
CURSOR	Usa l'interrupt BIOS 10H per posizionare il cursore.
BIOSDATA	Legge una stringa dalla ROM del BIOS e usa la funzione 09H dell' interrupt DOS 21H per visualizzare la stringa.
GETCH	Usa la funzione 08H dell'interrupt DOS 21H per leggere un carattere dalla tastiera.
SCHERMOF	Esempio di una routine di servizio di un interrupt. Usa l'interrupt BIOS 10H per leggere il modo video. Usa le funzioni 3CH, 3EH, 40H e 42H dell'interrupt DOS 21H per creare, scrivere, spostare un puntatore file e chiudere un file su disco.
SVCINST	Usa la funzione 25H dell'interrupt DOS 21H per assegnare una nuova routine di servizio all'interrupt 05H (PRINT SCREEN).
START	La routine principale che chiama tutte le altre. Usa la funzione 4CH dell'interrupt DOS per terminare il programma.

IL FILE BATCH PER CREARE IL PROGRAMMA DI DIMOSTRAZIONE

```
: assembla MAIN.ASM, genera MAIN.OBJ e MAIN.LST,
: comprensivo delle cross-reference.

masm main,,main/c;

: linka MAIN.OBJ, genera MAIN.EXE

link main;
```

Microsoft (R) Macro Assembler Version 5.00 1/3/89 13:23:22 Page 1-1

```
1                          page 66,132   ;page length 66 lines, width 132 lines
2
3                          MAIN.ASM 11.12.88
4
5 = 004C                   TERMINA   equ   4Ch          ;termina il programma
6
7                          includeCLRSCRN.ASM           ;altri programmi
8                C
9                C         CLRSCRN.ASM 11.12.88
10               C
11               C
12 = 0017        C         ATTRIB    equ   17h          ;clear screen attribute
13               C
14 0000          C         cseg      segment para public 'code'
15               C                   assume cs:cseg,ds:cseg,es:cseg,ss:cseg
16               C
17 0000          C         clrscrn   proc  far
18               C
19 0000 50       C                   push  ax           ;salva il contenuto di AX
20 0001 53       C                   push  bx           ;salva il contenuto di BX
21 0002 51       C                   push  cx           ;salva il contenuto di CX
22 0003 52       C                   push  dx           ;salva il contenuto di DX
23               C
24 0004 B8 0600  C                   mov   ax,0600h     ;06 = scorri, 00 = tutte le rige
25 0007 B7 17    C                   mov   bh,ATTRIB    ;attributo del video
26 0009 B9 0000  C                   mov   cx,0000h     ;riga 0, colonna 0
27 000C BA 184F  C                   mov   dx,184Fh     ;riga 24, colonna 79
28 000F CD 10    C                   int   10h          ;BIOS interrupt video
29               C
30 0011 5A       C                   pop   dx           ;ripristina il contenuto di DX
31 0012 59       C                   pop   cx           ;ripristina il contenuto di CX
32 0013 5B       C                   pop   bx           ;ripristina il contenuto di BX
33 0014 58       C                   pop   ax           ;ripristina il contenuto di AX
34 0015 CB       C                   ret                ;return
35               C
36 0016          C         clrscrn   endp
37               C
38 0016          C         cseg      ends
39               C
40

41                         include CORNICE.ASM ;
42               C
43               C         CORNICE.ASM 11.12.88
44               C         Controlla modo video corrente,
45               C         visualizza cornice sullo schermo del PC
46               C         (scrive direttamente alla memoria video).
47               C
48               C
49               C
```

```
50 = 000F          C   VIDMODE  equ   0Fh            ;legge il modo del video
51                 C
52 = 002A          C   BCHAR    equ   '*'            ;carattere del bordo
53 = 0020          C   FCHAR    equ   ' '            ;carattere della finestra
54 = 0007          C   BATTRIB  equ   07h            ;attributo del bordo
55                 C                                 ;(bianco su nero)
56 = 0017          C   FATTRIB  equ   17h            ;attributo della finestra
57                 C                                 ;(bianco su grigio o bianco su blu)
58                 C
59 = 0050          C   VIDCOLS  equ   80             ;assume 80 colonne x 25 righe
60 = 0019          C   VIDLINS  equ   25             ;
61                 C
62                 C
63                 C   ;         MEMORIA VIDEO MONO
64                 C
65 0000            C   vmono    segment at 0B000h    ;memoria video monocromatico
66 0000            C   vmono    ends
67                 C
68                 C
69                 C             MEMORIA VIDEO CGA
70                 C
71 0000            C   vcga     segment at 0B800h    ;mem. video colour graphics adaptor
72                 C
73 0000            C   vcga     ends
74                 C
75 0016            C   csegsegment para public 'code'
76                 C
77                 C
78                 C             LINEBEG
79                 C
80 0016            C   linebeg  proc  far
81                 C
82 0016 B8 072A    C            mov   ax,BCHAR+
                                (256*BATTRIB)        ;attributo e carattere del bordo
83 0019 B9 0050    C            mov   cx,VIDCOLS     ;tutte le colonne della riga 1
84 001C AB         C   lp1:     stosw                ;ax - es:[di++]
85 001D E2 FD      C            loop  lp1            ;loop, prossimo carattere
86 001F CB         C            ret                  ;return
87                 C
88 0020            C   linebeg  endp
89                 C
90                 C             MAIN.ASM 11.12.88
91                 C
92                 C
93 0020            C   lineint  proc  far
94                 C
95 0020 51         C            push  cx             ;salva contenuto del registro cx
96                 C
97 0021 B8 072A    C            mov   ax,BCHAR+
                                (256*BATTRIB)        ;chr e attr. del bordo a sinistra
98 0024 AB         C            stosw                ;ax - es:[di++]
99                 C
100 0025 B8 1720   C            mov   ax,FCHAR+
                                (256*FATTRIB)        ;chr e attr. della finestra
101 0028 B9 004E   C            mov   cx,VIDCOLS-2;una riga della finestra
102 002B AB        C   lp2:     stosw                ;ax - es:[di++]
```

```
103 002C E2 FD          C                loop  lp2           ;loop, prossimo carattere
04                      C
105 002E B8 072A        C                mov   ax,BCHAR+
                                         (256*BATTRIB)       ;chr e attr. del bordo a destra
106 0031 AB             C                stosw               ;ax - es:[di++]
107                     C
108 0032 59             C                pop   cx            ;ripristina contenuto CX
109 0033 CB             C                ret                 ;return
110                     C
111 0034                C   lineint      endp
112                     C
113 0034                C   cornice      proc  far
114                     C
115 0034 B4 0F          C                mov   ah,VIDMODE    ;legge il modo del video
116 0036 CD 10          C                int   10h           ;BIOS video interrupt
117                     C
118 0038 3C 02          C                cmp   al,02h        ;80 x 25 bianco e nero ?
119 003A 74 0B          C                je    beg1          ;si: è una CGA
120 003C 3C 03          C                cmp   al,03h        ;80 x 25 colore ?
121 003E 74 07          C                je    beg1          ;si: è una CGA
122 0040 3C 07          C                cmp   al,07h        ;80 x 25 monocromatico ?
123 0042 74 08          C                je    beg2          ;si: è una MDA
124 0044 EB 22 90       C                jmp   cexit         ;il modo video non va bene
125                     C
126 0047 B8 ---- R      C   beg1:        mov   ax,vcga       ;segmento di destinazione
127                     C                assume              es:vcga;informare MASM
128 004A EB 03          C                jmp   short beg3    ;
129                     C
130 004C B8 ---- R      C   beg2:        mov   ax,vmono      ;segmento di destinazione
131                     C                assume              es:vmono;informare MASM
132                     C
133 004F 8E C0          C   beg3:        mov   es,ax         ;destinazione è la memoria video
134 0051 33 FF          C                xor   di,di         ;destinazione offset
135                     C
136 0053 FC             C                cld                 ;auto-incremento SI e DI
137 0054 9A 0016 ---- R C                call  linebeg       ;prima riga
138                     C
139 0059 B9 0017        C                mov   cx,VIDLINS-2  ;numero di rige intermediate
140 005C 9A 0020 ---- R C   lp0:         call  lineint       ;righe intermediate
141 0061 E2 F9          C                loop  lp0           ;loop, prossima riga
142                     C
143 0063 9A 0016 ---- R C                call  linebeg       ;ultima riga
144                     C
145 0068 CB             C   cexit:       ret                 ;return
146                     C
147 0069                C   cornice      endp
148                     C
149 0069                C   cseg         ends
150                     C
151
152                         include      SUONO.ASM           ;
153                     C
154                     C   SUONO.ASM 11.12.88
155                     C
156                     C
157 = 0061              C   PPI          equ   61h           ;interfaccia parallela programmabile
```

```
158 = 0003          C   SPKR    equu        03h             ;bit di controllo altoparlante
159                 C
160 = 0042          C   PITCNT2             equ   42h       ;contatore 2
161 = 0043          C   PITCTRL             equ   43h       ;porta di controllo
162 = 00B6          C   DATCTRL             equ   0B6h      ;dati di controllo
163 = 0952          C   Hz500   equ         0952h           ;500 Hz: divisore caricato nel cont.
164 = 04A9          C   Hz1000  equ         04A9h           ;1000 Hz
165 = 0254          C   Hz2000  equ         0254h           ;2000 Hz
166                 C
167 = C000          C   RITARDO             equ   0C000h    ;valore di durata del loop di ritardo
168                 C
169 0069            C   csegsegment para public 'CODE'
170                 C
171 0069            C   suonov  proc        far
172                 C
173 0069 50         C           push        ax              ;
174 006A 51         C           push        cx              ;
175                 C
176 006B 50         C           push        ax              ;salva il divisore
177 006C B0 B6      C           mov         al,DATCTRL      ;dati di controllo
178 006E E6 43      C           out         PITCTRL,al      ;scrive porta di controllo
179 0070 58         C           pop         ax              ;recupera divisore
180                 C
181 0071 E6 42      C           out         PITCNT2,al      ;scrive divisore byte meno significat.
182 0073 86 E0      C           xchg        ah,al           ;
183 0075 E6 42      C           out         PITCNT2,al      ;scrive divisore byte più significat.
184                 C
185 0077 E4 61      C           in          al,PPI          ;leggere i bit di controllo altoparlante
186 0079 0C 03      C           or          al,SPKR         ;abilita l'altoparlante
187 007B E6 61      C           out         PPI,al          ;scrive
188                 C
189 007D B9 C000    C           mov         cx,RITARDO      ;
190 0080 E2 FE      C   suonol: loop        suonol          ;attende
191                 C
192 0082 E4 61      C           in          al,PPI          ;legge i bit di controllo altoparlante
193 0084 24 FC      C           and         al,NOT SPKR     ;disabilita l'altoparlante
194 0086 E6 61      C           out         PPI,al          ;scrive
195                 C
196 0088 59         C           pop         cx              ;
197 0089 58         C           pop         ax              ;
198 008A CB         C           ret                         ;return
199                 C
200 008B            C   suonov  endp
201                 C
202 008B            C   suono   proc        far
203                 C
204 008B 50         C           push        ax              ;salva contenuto AX
205 008C B8 0254    C           mov         ax,Hz2000       ;2000 Hz
206 008F 9A 0069.R  C           call        suonov          ;suona
207 0094 B8 04A9    C           mov         ax,Hz1000       ;1000 Hz
208 0097 9A 0069.R  C           call        suonov          ;suona
209 009C B8 0952    C           mov         ax,Hz500        ;500 Hz
210 009F 9A 0069.R  C           call        suonov          ;suona
211 00A4 58         C           pop         ax              ;ripristina contenuto AX
212 00A5 CB         C           ret                         ;return
213                 C
```

```
214 00A6                C      suono     endp
215                     C
216 00A6                C      cseg      ends
217                     C
218
219                            include   CURSOR.ASM
220                     C
221                     C      CURSOR.ASM 11.12.88
222                     C
223                     C
224 00A6                C      csegsegment para public 'CODE'
225                     C      assume cs:cseg,ds:dseg,es:dseg,ss:sseg
226                     C
227 00A6                C      cursor    proc   far
228                     C
229 00A6 50             C                push   ax           ;salva il contenuto di AX
230 00A7 53             C                push   bx           ;salva il contenuto di BX
231 00A8 52             C                push   dx           ;salva il contenuto di DX
232                     C
233 00A9 B7 00          C                mov    bh,00h       ;pagina video 0
234 00AB 8B D0          C                mov    dx,ax        ;DH = riga, DL = colonna
235 00AD B4 02          C                mov    ah,02h       ;posizionamento cursore
236 00AF CD 10          C                int    10h          ;BIOS video interrupt
237                     C
238 00B1 5A             C                pop    dx           ;ripristina il contenuto di DX
239 00B2 5B             C                pop    bx           ;ripristina il contenuto di BX
240 00B3 58             C      pop       ax                  ;ripristina il contenuto di AX
241 00B4 CB             C                ret                 ;return
242                     C
243 00B5                C      cursor    endp
244                     C
245 00B5                C      cseg      ends
246                     C
247
248                            include   BIOSDATA.ASM        ;
249                     C
250                     C      BIOSDATA.ASM 11.12.88
251                     C
252 = F000              C      SEGBIOS   equ    0F000h       ;segmento del BIOS
253 = FFF5              C      VERDAT    equ    0FFF5h       ;offset della data del BIOS
254 = 0009              C      STRDISP   equ    09h          ;visualizza una stringa
255 = 000D              C      CR        equ    0Dh          ;carriage return
256 = 000A              C      LF        equ    0Ah          ;line feed
257                     C
258                     C      ; copia la data BIOS in questa string
259                     C
260 0000                C      dseg      segment para public 'DATA'
261 0000 6D6D2F67672F61C       buftmp    db     "mm/gg/aa",CR,LF,'$';
26261 0D 0A 24          C
263 000B                C      dseg      ends
264                     C
265 00B5                C      csegsegment para public 'CODE'
266                     C      assume cs:cseg,ds:dseg,es:dseg,ss:sseg
267                     C
268 00B5                C      biosdata proc far
269                     C
```

```
Microsoft (R) Macro Assembler Version 5.00 1/3/89 13:23:22 Page 1-6

270 00B5 50            C            push  ax              ;salva i registri usati
271 00B6 51            C            push  cx              ;
272 00B7 52            C            push  dx              ;
273 00B8 56            C            push  si              ;
274 00B9 57            C            push  di              ;
275 00BA 06            C            push  es              ;
276                    C
277 00BB 1E            C            push  ds              ;copia ds in es
278 00BC 07            C            pop   es              ;
279 00BD BF 0000 R     C            mov   di,offset buftmp;es:di punta alla stringa
280 00C0 1E            C            push  ds              ;salva ds
281 00C1 B8 F000       C            mov   ax,SEGBIOS      ;
282 00C4 8E D8         C            mov   ds,ax           ;
283 00C6 BE FFF5       C            mov   si,VERDAT       ;ds:si punta alla data BIOS
284 00C9 B9 0008       C            mov   cx,8            ;copia 8 carattere
285 00CC FC            C            cld                   ;auto-incremento
286 00CD F3/ A4        C            rep   movsb           ;ds:[si++] - es:[di++], djnz cx
287 00CF 1F            C            pop   ds              ;ripristа ds
288                    C
289 00D0 BA 0000 R     C            mov   dx,offset buftmp;ds:dx punta alla stringa
290 00D3 B4 09         C            mov   ah,STRDISP      ;visualizza la stringa
291 00D5 CD 21         C            int   21h             ;funzione DOS
292                    C
293 00D7 07            C            pop   es              ;ripristina i registri
294 00D8 5F            C            pop   di              ;
295 00D9 5E            C            pop   si              ;
296 00DA 5A            C            pop   dx              ;
297 00DB 59            C            pop   cx              ;
298 00DC 58            C            pop   ax              ;
299 00DD CB            C            ret                   ;
300                    C
301 00DE               C  biosdata endp
302                    C
303 00DE               C  cseg      ends
304                    C
305
306                       include   GETCH.ASM             ;
307                    C
308                    C  GETCH.ASM 11.12.88
309                    C
310                    C
311 = 0008             C  GETCHAR   equ   08h             ;legge un carattere dalla tastiera,
312                    C                                  ;senza eco
313                    C
314 00DE               C  csegsegment para public 'code'
315                    C
316 00DE               C  getch     proc  far
317                    C
318 00DE B4 08         C            mov   ah,GETCHAR      ;input dalla tastiera senza eco
319 00E0 CD 21         C            int   21h             ;funzione DOS
320 00E2 CB            C            ret                   ;return
321                    C
322 00E3               C  getch     endp
323                    C
324 00E3               C  cseg      ends
325                    C
```

```
326
327                                 include     SCHERMOF.ASM    ;
328                       C                     public inhand
329                       C
330                       C              SCHERMOF.ASM 11.12.88
331                       C
332                       C              Salva il contenuto dello schermo sul file
333                       C              SCHERMO.PIC nella directory corrente
334                       C              premendo SHIFT-PRINTSCREEN.
335                       C
336                       C              modo 1 (25 righe x 40 colonne, colore)
337                       C
338                       C              bit di attributo:              7 6 5 4      3 2 1 0
339                       C              (1 byte/char) i r g b i r g b
340                       C                                             background   foreground
341                       C
342                       C              pagina video 0 = b8000 to b87d0
343                       C              pagina video 1 = b8800 to b8fa0, etc
344                       C              pagina video 15 = bf800 to bffa0
345                       C
346                       C              modo 3 (25 righe x 80 colonne, colore)
347                       C
348                       C              bit di attributo:              7 6 5 4      3 2 1 0
349                       C              (1 byte/char) i r g b i r g b
350                       C                                             background foreground
351                       C
352                       C              pagina video 0 = b8000 to b8fa0
353                       C              pagina video 1 = b9000 to b9fa0, etc
354                       C              pagina video 7 = bf000 to bffa0
355                       C
356                       C              modo 4 (200 righe x 320 colonne, colore)
357                       C
358                       C              bit dei pixel:                 7 6 5 4      3 2 1 0
359                       C              numero del pixel:0 1 2 3
360                       C              2-bit colore:                  x x x x      x x x x
361                       C              (00 = background, 01, 10, 11 = foreground)
362                       C
363                       C              pagina video 0, righe 0,2,.. = b8000 - b9f3f
364                       C              pagina video 0, righe 1,3,.. = ba000 - bbf3f
365                       C              pagina video 1, righe 0,2,.. = bc000 - bdf3f
366                       C              pagina video 1, righe 1,3,.. = be000 - bff3f
367                       C
368 = 000F                C      GETVMOD   equ   0Fh       ;legge modo video
369 = 0025                C      SETINTR   equ   25h       ;assegnare vettore di interrupt
370 = 003C                C      CREATE    equ   3Ch       ;creare file handle
371 = 003E                C      CLOSE     equ   3Eh       ;chiudere file handle
372 = 0040                C      WRITE     equ   40h       ;scrivere file handle
373 = 0042                C      SEEK      equ   42h       ;muovere file pointer
374                       C
375 00E3                  C      csegsegment para public 'CODE'
376                       C
377                       C      ; NOTA: Dato che l'interrupt puo' avvenire in qualunque momento,
378                       C      i dati devono essere nel CODE segment.***
379                       C
380 00E3 53434845524D4F C        path        db    "SCHERMO.PIC",00 ;nome ASCIIZ
                                                   del file per immagine schermo
```

```
3812E 50 49 43 00          C
382 00EF B800              C    bufseg        dw    0B800h        ;indirizzo memoria video CGA
383 00F1 00                C    inhand        db    00h           ;inhand = 1
                                                                   quando all'interno di schermof
384                        C
385 00F2                   C    schermof proc       far
386                        C
387 00F2 FB                C                  sti                 ;** riabilitare subito gli interrupt **
388 00F3 50                C                  push  ax            ;interrupt handler di PRINTSCREEN
389 00F4 53                C                  push  bx            ;
390 00F5 51                C                  push  cx            ;
391 00F6 52                C                  push  dx            ;
392 00F7 56                C                  push  si            ;
393 00F8 57                C                  push  di            ;
394 00F9 1E                C                  push  ds            ;
395 00FA 06                C                  push  es            ;
396                        C
397 00FB 0E                C                  push  cs            ;
398 00FC 1F                C                  pop   ds            ;ds = cs (tutto nello stesso segmento)
399                        C
400 00FD 2E:803E00F1R00 C                     cmp   inhand,00h    ;inhand = 0 ?
401 0103 75 50             C                  jnz   quit          ;si: salta (gia' nell'interrupt handler)
402 0105 2E: FE 06 00F1 R C                   inc   inhand        ;setta inhand = 1
403                        C
404 010A BA 00E3 R         C                  mov   dx,offset path ;ds:dx = pathname del file da creare
405 010D B9 0000           C                  mov   cx,0000h      ;cx = attributo file
                                                                   = leggere/scrivere normale
406 0110 B4 3C             C                  mov   ah,CREATE     ;creare file handle
407 0112 CD 21             C                  int   21h           ;funzione DOS
408                        C
409 0114 50                C                  push  ax            ;salvare file handle
410                        C
411 0115 8B D8             C                  mov   bx,ax
412 0117 33 C9             C                  xor   cx,cx         ;cx:dx = offset dall'inizio del file
413 0119 33 D2             C                  xor   dx,dx         ;
414 011B 32 C0             C                  xor   al,al         ;andare all'inizio del file + offset
415 011D B4 42             C                  mov   ah,SEEK       ;muovere file pointer
416 011F CD 21             C                  int   21h           ;funzione DOS
417                        C
418 0121 B4 0F             C                  mov   ah,GETVMOD ;leggere modo video
419 0123 CD 10             C                  int   10h           ;funzione BIOS video
420                        C
421 0125 B9 07D0           C                  mov   cx,2000       ;(25 righe x 40 col)/1
                                                                   char+attrib per 2 byte
422 0128 3C 01             C                  cmp   al,1          ;modo video 1 (25 x 40 colore) ?
423 012A 74 11             C                  je    wrfile        ;si
424                        C
425 012C B9 0FA0           C                  mov   cx,4000       ;(25 righe x 80 col)/1
                                                                   char+attrib per 2 byte
426 012F 3C 03             C                  cmp   al,3          ;modo video 3 (25 x 80 colore) ?
427 0131 74 0A             C                  je    wrfile        ;si
428                        C
429 0133 B9 3F40           C                  mov   cx,16192      ;(200 righe x 320 col)/4
                                                                   dot+attrib per byte
430                        C                                      ;(+192 byte inusati alla fine
                                                                   delle righe 0,2,..)
```

```
431 0136 3C 04          C                cmp   al,4          ;modo video 4 (200 x 320 colore) ?
432 0138 74 03          C                je    wrfile        ;si
433                     C
434 013A B9 0FA0        C                mov   cx,4000       ;default: modo video
                                                             3 (25 x 80 colore)
435                     C
436                     C                                    ;cx = numero di byte da scrivere
437 013D 5B             C   wrfile:      pop   bx            ;ripristinare file handle
438                     C
439 013E 53             C                push  bx            ;salvare file handle
440 013F 2E: 8E 1E 00EF RC               mov   ds,bufseg     ;
441 0144 33 D2          C                xor   dx,dx         ;ds:dx = buffer da cui scrivere
442 0146 B4 40          C                mov   ah,WRITE      ;scrivere al file handle
443 0148 CD 21          C                int   21h           ;funzione DOS
444 014A 5B             C                pop   bx            ;ripristinare file handle
445                     C
446 014B B4 3E          C                mov   ah,CLOSE      ;chiudere file handle
447 014D CD 21          C                int   21h           ;funzione DOS
448                     C
449                     C                                    ;DS qui e' sbagliato, quindi usare CS
450 014F 2E:C60600F1R00 C                mov   cs:inhand,00h ;inhand = 0
                                                             (interrupt handler terminata)
451                     C
452 0155 07             C   quit:        pop   es            ;
453 0156 1F             C                pop   ds            ;
454 0157 5F             C                pop   di            ;
455 0158 5E             C                pop   si            ;
456 0159 5A             C                pop   dx            ;
457 015A 59             C                pop   cx            ;
458 015B 5B             C                pop   bx            ;
459 015C 58             C                pop   ax            ;
460 015D CF             C                iret                ;ritorno dall'interrupt
461                     C
462 015E                C   schermof endp
463                     C
464                     C
465                     C
466                     C   SVCINST
467                     C
468 015E                C   svcinst      proc  far
469                     C
470 015E 50             C                push  ax            ;
471 015F 52             C                push  dx            ;
472 0160 1E             C                push  ds            ;
473 0161 0E             C                push  cs            ;copia cs in ds
474 0162 1F             C                pop   ds            ;
475 0163 BA 00F2 R      C                mov   dx,offset schermof ;ds:dx = indirizzo di schermof
476 0166 B4 25          C                mov   ah,SETINTR    ;assegnare vettore di interrupt
477 0168 B0 05          C                mov   al,05h        ;PRINTSCREEN interrupt
478 016A CD 21          C                int   21h           ;funzione DOS
479 016C 1F             C                pop   ds            ;
480 016D 5A             C                pop   dx            ;
481 016E 58             C                pop   ax            ;
482 016F CB             C                ret                 ;
483                     C
484 0170                C   svcinst      endp
```

```
Microsoft (R) Macro Assembler Version 5.00 1/3/89 13:23:22 Page 1-10

485                     C
486 0170                C       cseg        ends
487                     C
488
489
490 0170                        csegsegment para public 'CODE'
491                             assume cs:cseg,ds:dseg,es:dseg,ss:sseg
492
493 0170                        start       proc    near
494
495 0170 B8 ---- R              mov         ax,dseg         ;punta al segmento dati
496 0173 8E D8                  mov         ds,ax           ;
497 0175 8E C0                              mov     es,ax   ;
498
499 0177 9A 015E ---- R         call        svcinst         ;installa funzione per salvare video
500 017C 9A 0000 ---- R         call        clrscrn         ;cancella lo schermo
501 0181 9A 0034 ---- R         call        cornice         ;visualizza la cornice
502 0186 9A 008B ---- R         call        suono           ;bip
503 018B B8 0C24                mov         ax,0C24h        ;riga 12, colonna 36
504 018E 9A 00A6 ---- R         call        cursor          ;posiziona il cursore
505 0193 9A 00B5 ---- R         call        biosdata        ;visualizza l'area dati del BIOS
506 0198 9A 00DE ---- R         call        getch           :attende che un tasto sia premuto
507 019D B8 1901                mov         ax,1901h        ;riga 25, colonna 01
508 01A0 9A 00A6 ---- R         call        cursor          ;posiziona il cursore
509
510 01A5 B0 00                  mov         al,00h          ;codice di ritorno
511 01A7 B4 4C                  mov         ah,TERMINA      ;termina il programma
512 01A9 CD 21                  int         21h             ;funzione MS-DOS
513
514 01AB                        start       endp
515
516 01AB                        cseg        ends
517
518
519                                         STACK
520
521 0000                        sseg        segment         para stack 'STACK'
522
523 0000 007F[                  stkend      dw   127 dup (?)  ;lo stack termina qui
524          ????
525                     ]
526
527 00FE ????                   stktop      dw   ?          ;lo stack inizia qui
528
529 0100                        sseg        ends
530                                         end  start      ;definisce il punto di entrata
```

```
Microsoft (R) Macro Assembler Version 5.00 1/3/89 13:23:22
Symbols-1


Segments and Groups:

N a m e ..................................Length        Align          Combine Class

CSEG......................................01AB          PARA           PUBLIC   'CODE'
DSEG......................................000B          PARA           PUBLIC   'DATA'
SSEG......................................0100          PARA           STACK    'STACK'
VCGA......................................0000          AT             B800
VMONO.....................................0000          AT             B000

Symbols:

N a m e ..................................Type          Value          Attr

ATTRIB ...................................NUMBER        0017

BATTRIB ..................................NUMBER        0007
BCHAR ....................................NUMBER        002A
BEG1 .....................................L NEAR        0047           CSEG
BEG2 .....................................L NEAR        004C           CSEG
BEG3 .....................................L NEAR        004F           CSEG
BIOSDATA ...........................                    F PROC         00B5     CSEGLength = 0029
BUFSEG....................................L WORD        00EF           CSEG
BUFTMP....................................L BYTE        0000           DSEG

CEXIT ....................................L NEAR        0068           CSEG
CLOSE ....................................NUMBER        003E
CLRSCRN...................................F PROC        0000           CSEG     Length = 0016
CORNICE...................................F PROC        0034           CSEG     Length = 0035
CR........................................NUMBER        000D
CREATE....................................NUMBER        003C
CURSOR ...................................F PROC        00A6           CSEG     Length = 000F

DATCTRL ..................................NUMBER        00B6

FATTRIB ..................................NUMBER        0017
FCHAR.....................................NUMBER        0020

GETCH.....................................F PROC        00DE           CSEG     Length = 0005
GETCHAR ..................................NUMBER        0008
GETVMOD ...........................                     NUMBER         000F

HZ1000....................................NUMBER        04A9
HZ2000....................................NUMBER        0254
HZ500.....................................NUMBER        0952

INHAND ...................................L BYTE        00F1           CSEG     Global

LF .......................................NUMBER        000A
LINEBEG...................................F PROC        0016           CSEG     Length = 000A
LINEINT ..................................F PROC        0020           CSEG     Length = 0014
LP0 ......................................L NEAR        005C           CSEG
LP1 ......................................L NEAR        001C           CSEG
LP2 ......................................L NEAR        002B           CSEG
```

```
Microsoft (R) Macro Assembler Version 5.00 1/3/89 13:23:22
Symbols-2

PATH .....................................L BYTE     00E3    CSEG
PITCNT2 ..................................NUMBER     0042
PITCTRL ..................................NUMBER     0043
PPI ......................................NUMBER     0061

QUIT .....................................L NEAR     0155    CSEG

RITARDO ..................................NUMBER     C000

SCHERMOF .................................F PROC     00F2    CSEG    Length = 006C
SEEK .....................................NUMBER     0042
SEGBIOS ..................................NUMBER     F000
SETINTR ..................................NUMBER     0025
SPKR .....................................NUMBER     0003
START ....................................N PROC     0170    CSEG    Length = 003B
STKEND ...................................L WORD     0000    SSEG    Length = 007F
STKTOP ...................................L WORD     00FE    SSEG
STRDISP ..................................NUMBER     0009
SUONO ....................................F PROC     008B    CSEG    Length = 001B
SUONO1 ...................................L NEAR     0080    CSEG
SUONOV ...................................F PROC     0069    CSEG    Length = 0022
SVCINST ..................................F PROC     015E    CSEG    Length = 0012

TERMINA ..................................NUMBER     004C

VERDAT ...................................NUMBER     FFF5
VIDCOLS ..................................NUMBER     0050
VIDLINS ..................................NUMBER     0019
VIDMODE ..................................NUMBER     000F

WRFILE ...................................L NEAR     013D    CSEG
WRITE ....................................NUMBER     0040

@FILENAME ................................TEXT main


518 Source Lines
525 TotalLines
68 Symbols

50396 + 363476 Bytes symbol space free

0 Warning Errors
0 Severe Errors
```

APPENDICI

APPENDICE A

TABELLA DI CONVERSIONE ESADECIMALE

HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	00	000
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	256	4096
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	512	8192
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	768	12288
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	1024	16384
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	1280	20480
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	1536	24576
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	1792	28672
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	2048	32768
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	2304	36864
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	2560	40960
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	2816	45056
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	3072	49152
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	3328	53248
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	3584	57344
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	3840	61440

5		4		3		2		1		0	
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0	0	0	0	0
1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15

APPENDICE B

TABELLA DI CONVERSIONE ASCII

HEX	MSD	0	1	2	3	4	5	6	7
LSD	BITS	000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SPACE	0	@	P	—	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	″	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	′	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	--
D	1101	CR	GS	−	=	M	]	m	}
E	1110	SO	RS	.	>	N	^	n	~
F	1111	SI	US	/	?	O	←	o	DEL

SIMBOLI ASCII

DLE – Uscita dal ciclo dati
DC – Controllo dispositivo
NAK – Riconoscimento negativo
SYN – Rinvio sincronizzato
ETB – Fine blocco trasmissione
CAN – Cancellare
EM – Fine del medium
SUB – Sostituto
ESC – Escape
FS – Separatore di file
GS – Separatore di gruppo
RS – Separatore di record
US – Separatore di unità
SP – Spazio (blank)
DEL – Delete
NUL – Nullo

SOH – Inizio interazione
STX – Inizio testo
ETX – Fine testo
EOT – Fine della trasmissione
ENQ – Richiesta
ACK – Riconoscimento
BEL – Campanello
BS – Spazio indietro
HT – Tabulazione orizzontale
LF – Avanzamento linea
VT – Tabulazione verticale
FF – Avanzamento pagina
CR – Ritorno carrello
SO – Shift out
SI – Shift in

APPENDICE C

TABELLA DI CONVERSIONE DA DECIMALE A BCD

DECIMAL	BCD	DEC	BCD	DEC	BCD
0	0000	10	00010000	91	10010000
1	0001	11	00010001	91	10010001
2	0010	12	00010010	92	10010010
3	0011	13	00010011	93	10010011
4	0100	14	00010100	94	10010100
5	0101	15	00010101	95	10010101
6	0110	16	00010110	96	10010110
7	0111	17	00010111	97	10010111
8	1000	18	00011000	98	10011000
9	1001	19	00011001	99	10011001

APPENDICE D

SET DI ISTRUZIONI 8086/8088

AAA	**AAA** (no operands) ASCII adjust for addition			Flags O D I T S Z A P C U UU X U X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	4	—	1	AAA

AAD	**AAD** (no operands) ASCII adjust for division			Flags O D I T S Z A P C U X X U X U
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	60	—	2	AAD

AAM	**AAM** (no operands) ASCII adjust for multiply			Flags O D I T S Z A P C U X X U X U
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	83	—	1	AAM

AAS	**AAS** (no operands) ASCII adjust for subtraction			Flags O D I T S Z A P C U U U X U X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	4	—	1	AAS

ADC	**ADC** destination,source Add with carry			Flags O D I T S Z A P C X X X X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, register	3	—	2	ADC AX, SI
register, memory	9+EA	1	2-4	ADC DX, BETA [SI]
memory, register	16+EA	2	2-4	ADC ALPHA [BX] [SI], DI
register, immediate	4	—	3-4	ADC BX, 256
memory, immediate	17+EA	2	3-6	ADC GAMMA, 30H
accumulator, immediate	4	—	2-3	ADC AL, 5

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

ADD

ADD destination,source
Addition

Flags	O	D	I	T	S	Z	A	P	C
	X				X	X	X	X	X

Operands	Clocks	Transfers*	Bytes	Coding Example
register, register	3	—	2	ADD CX, DX
register, memory	9+EA	1	2-4	ADD DI, [BX].ALPHA
memory, register	16+EA	2	2-4	ADD TEMP, CL
register, immediate	4	—	3-4	ADD CL, 2
memory, immediate	17+EA	2	3-6	ADD ALPHA, 2
accumulator, immediate	4	—	2-3	ADD AX, 200

AND

AND destination,source
Logical and

Flags	O	D	I	T	S	Z	A	P	C
	0				X	X	U	X	0

Operands	Clocks	Transfers*	Bytes	Coding Example
register, register	3	—	2	AND AL,BL
register, memory	9+EA	1	2-4	AND CX,FLAG_WORD
memory, register	16+EA	2	2-4	AND ASCII [DI],AL
register, immediate	4	—	3-4	AND CX,0F0H
memory, immediate	17+EA	2	3-6	AND BETA, 01H
accumulator, immediate	4	—	2-3	AND AX, 01010000B

CALL

CALL target
Call a procedure

Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Examples
near-proc	19	1	3	CALL NEAR_PROC
far-proc	28	2	5	CALL FAR_PROC
memptr 16	21+EA	2	2-4	CALL PROC_TABLE [SI]
regptr 16	16	1	2	CALL AX
memptr 32	37+EA	4	2-4	CALL [BX].TASK [SI]

CBW

CBW (no operands)
Convert byte to word

Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	CBW

CLC

CLC (no operands)
Clear carry flag

Flags	O	D	I	T	S	Z	A	P	C
									0

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	CLC

CLD

CLD (no operands)
Clear direction flag

Flags	O	D	I	T	S	Z	A	P	C
		0							

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	CLD

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

CLI

CLI (no operands)
Clear interrupt flag

Flags	O	D	I	T	S	Z	A	P	C
			0						

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	CLI

CMC

CMC (no operands)
Complement carry flag

Flags	O	D	I	T	S	Z	A	P	C
									X

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	CMC

CMP

CMP destination,source
Compare destination to source

Flags	O	D	I	T	S	Z	A	P	C
	X				X	X	X	X	X

Operands	Clocks	Transfers*	Bytes	Coding Example
register, register	3	—	2	CMP BX, CX
register, memory	9+EA	1	2-4	CMP DH, ALPHA
memory, register	9+EA	1	2-4	CMP [BP+2], SI
register, immediate	4	—	3-4	CMP BL, 02H
memory, immediate	10+EA	1	3-6	CMP [BX].RADAR[DI], 3420H
accumulator, immediate	4	—	2-3	CMP AL, 00010000B

CMPS

CMPS dest-string,source-string
Compare string

Flags	O	D	I	T	S	Z	A	P	C
	X				X	X	X	X	X

Operands	Clocks	Transfers*	Bytes	Coding Example
dest-string, source-string	22	2	1	CMPS BUFF1, BUFF2
(repeat) dest-string, source-string	9+22/rep	2/rep	1	REPE CMPS ID, KEY

CWD

CWD (no operands)
Convert word to doubleword

Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	5	—	1	CWD

DAA

DAA (no operands)
Decimal adjust for addition

Flags	O	D	I	T	S	Z	A	P	C
	X				X	X	X	X	X

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	4	—	1	DAA

DAS

DAS (no operands)
Decimal adjust for subtraction

Flags	O	D	I	T	S	Z	A	P	C
	U				X	X	X	X	X

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	4	—	1	DAS

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

DEC	**DEC** destination Decrement by 1			Flags O D I T S Z A P C X X X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg16	2	—	1	DEC AX
reg8	3	—	2	DEC AL
memory	15+EA	2	2-4	DEC ARRAY [SI]

DIV	**DIV** source Division, unsigned			Flags O D I T S Z A P C U U U U U U
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg8	80-90	—	2	DIV CL
reg16	144-162	—	2	DIV BX
mem8	(86-96) +EA	1	2-4	DIV ALPHA
mem16	(150-168) +EA	1	2-4	DIV TABLE [SI]

ESC	**ESC** external-opcode,source Escape			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
immediate, memory	8+EA	1	2-4	ESC 6,ARRAY [SI]
immediate, register	2	—	2	ESC 20,AL

HLT	**HLT** (no operands) Halt			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	2	—	1	HLT

IDIV	**IDIV** source Integer division			Flags O D I T S Z A P C U U U U U U
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg8	101-112	—	2	IDIV BL
reg16	165-184	—	2	IDIV CX
mem8	(107-118) +EA	1	2-4	IDIV DIVISOR_BYTE [SI]
mem16	(171-190) +EA	1	2-4	IDIV [BX].DIVISOR_WORD

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

IMUL

IMUL source
Integer multiplication

Flags	O	D	I	T	S	Z	A	P	C
	X				U	U	U	U	X

Operands	Clocks	Transfers*	Bytes	Coding Example
reg8	80-98	—	2	IMUL CL
reg16	128-154	—	2	IMUL BX
mem8	(86-104) + EA	1	2-4	IMUL RATE BYTE
mem16	(134-160) + EA	1	2-4	IMUL RATE WORD [BP] [DI]

IN

IN accumulator,port
Input byte or word

Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Example
accumulator, immed8	10	1	2	IN AL, 0FFEAH
accumulator, DX	8	1	1	IN AX, DX

INC

INC destination
Increment by 1

Flags	O	D	I	T	S	Z	A	P	C
	X				X	X	X	X	

Operands	Clocks	Transfers*	Bytes	Coding Example
reg16	2	—	1	INC CX
reg8	3	—	2	INC BL
memory	15 + EA	2	2-4	INC ALPHA [DI] [BX]

INT

INT interrupt-type
Interrupt

Flags	O	D	I	T	S	Z	A	P	C
			0	0					

Operands	Clocks	Transfers*	Bytes	Coding Example
immed8 (type = 3)	52	5	1	INT 3
immed8 (type ≠ 3)	51	5	2	INT 67

INTR†

INTR (external maskable interrupt)
Interrupt if INTR and IF=1

Flags	O	D	I	T	S	Z	A	P	C
			0	0					

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	61	7	N/A	N/A

INTO

INTO (no operands)
Interrupt if overflow

Flags	O	D	I	T	S	Z	A	P	C
			0	0					

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	53 or 4	5	1	INTO

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

†INTR is not an instruction; it is included only for timing information.

IRET	**IRET** (no operands) Interrupt Return			Flags O D I T S Z A P C R R R R R R R R R
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	24	3	1	IRET

JA/JNBE	**JA/JNBE** short-label Jump if above/Jump if not below nor equal			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JA ABOVE

JAE/JNB	**JAE/JNB** short-label Jump if above or equal/Jump if not below			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JAE ABOVE EQUAL

JB/JNAE	**JB/JNAE** short-label Jump if below/Jump if not above nor equal			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JB BELOW

JBE/JNA	**JBE/JNA** short-label Jump if below or equal/Jump if not above			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JNA NOT ABOVE

JC	**JC** short-label Jump if carry			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JC CARRY SET

JCXZ	**JCXZ** short-label Jump if CX is zero			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	18 or 6	—	2	JCXZ COUNT DONE

JE/JZ	**JE/JZ** short-label Jump if equal/Jump if zero			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JZ ZERO

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

JG/JNLE	**JG/JNLE** short-label Jump if greater/Jump if not less nor equal			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JG GREATER

JGE/JNL	**JGE/JNL** short-label Jump if greater or equal/Jump if not less			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JGE GREATER_EQUAL

JL/JNGE	**JL/JNGE** short-label Jump if less/Jump if not greater nor equal			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JL LESS

JLE/JNG	**JLE/JNG** short-label Jump if less or equal/Jump if not greater			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JNG NOT_GREATER

JMP	**JMP** target Jump			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	15	—	2	JMP SHORT
near-label	15	—	3	JMP WITHIN_SEGMENT
far-label	15	—	5	JMP FAR_LABEL
memptr16	18+EA	1	2-4	JMP [BX].TARGET
regptr16	11	—	2	JMP CX
memptr32	24+EA	2	2-4	JMP OTHER.SEG [SI]

JNC	**JNC** short-label Jump if not carry			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JNC NOT_CARRY

JNE/JNZ	**JNE/JNZ** short-label Jump if not equal/Jump if not zero			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JNE NOT_EQUAL

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

JNO	JNO short-label Jump if not overflow			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JNO NO_OVERFLOW

JNP/JPO	JNP/JPO short-label Jump if not parity/Jump if parity odd			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JPO ODD_PARITY

JNS	JNS short-label Jump if not sign			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JNS POSITIVE

JO	JO short-label Jump if overflow			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JO SIGNED_OVRFLW

JP/JPE	JP/JPE short-label Jump if parity/Jump if parity even			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JPE EVEN_PARITY

JS	JS short-label Jump if sign			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JS NEGATIVE

LAHF	LAHF (no operands) Load AH from flags			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	4	—	1	LAHF

LDS	LDS destination,source Load pointer using DS			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers**	**Bytes**	**Coding Example**
reg16, mem32	16+EA	2	2-4	LDS SI,DATA.SEG [DI]

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

LEA	LEA destination,source Load effective address			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg16, mem16	2+EA	—	2-4	LEA BX, [BP] [DI]

LES	LES destination,source Load pointer using ES			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg16, mem32	16+EA	2	2-4	LES DI, [BX].TEXT_BUFF

LOCK	LOCK (no operands) Lock bus			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	2	—	1	LOCK XCHG FLAG,AL

LODS	LODS source-string Load string			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
source-string (repeat) source-string	12 9+13/rep	1 1/rep	1 1	LODS CUSTOMER_NAME REP LODS NAME

LOOP	LOOP short-label Loop			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	17/5	—	2	LOOP AGAIN

LOOPE/LOOPZ	LOOPE/LOOPZ short-label Loop if equal/Loop if zero			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	18 or 6	—	2	LOOPE AGAIN

LOOPNE/LOOPNZ	LOOPNE/LOOPNZ short-label Loop if not equal/Loop if not zero			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	19 or 5	—	2	LOOPNE AGAIN

NMI†	NMI (external nonmaskable interrupt) Interrupt if NMI = 1			Flags O S I T S Z A P C 0 0
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	50	5	N/A	N/A

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

†NMI is not an instruction; it is included only for timing information.

MOV	MOV destination,source Move			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
memory, accumulator	10	1	3	MOV ARRAY [SI], AL
accumulator, memory	10	1	3	MOV AX, TEMP_RESULT
register, register	2	—	2	MOV AX,CX
register, memory	8+EA	1	2-4	MOV BP, STACK_TOP
memory, register	9+EA	1	2-4	MOV COUNT [DI], CX
register, immediate	4	—	2-3	MOV CL, 2
memory, immediate	10+EA	1	3-6	MOV MASK [BX] [SI], 2CH
seg-reg, reg16	2	—	2	MOV ES, CX
seg-reg, mem16	8+EA	1	2-4	MOV DS, SEGMENT_BASE
reg16, seg-reg	2	—	2	MOV BP, SS
memory, seg-reg	9+EA	1	2-4	MOV [BX].SEG_SAVE, CS

MOVS	MOVS dest-string,source-string Move string			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
dest-string, source-string	18	2	1	MOVS LINE EDIT_DATA
(repeat) dest-string, source-string	9+17/rep	2/rep	1	REP MOVS SCREEN, BUFFER

MOVSB/MOVSW	MOVSB/MOVSW (no operands) Move string (byte/word)			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	18	2	1	MOVSB
(repeat) (no operands)	9+17/rep	2/rep	1	REP MOVSW

MUL	MUL source Multiplication, unsigned			Flags O D I T S Z A P C X U U U U X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg8	70-77	—	2	MUL BL
reg16	118-133	—	2	MUL CX
mem8	(76-83) +EA	1	2-4	MUL MONTH [SI]
mem16	(124-139) +EA	1	2-4	MUL BAUD_RATE

NEG	NEG destination Negate			Flags O D I T S Z A P C X X X X X 1*
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register	3	—	2	NEG AL
memory	16+EA	2	2-4	NEG MULTIPLIER

*0 if destination = 0

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

NOP	**NOP** (no operands) No Operation			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	3	—	1	NOP

NOT	**NOT** destination Logical not			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register	3	—	2	NOT AX
memory	16+EA	2	2-4	NOT CHARACTER

OR	**OR** destination,source Logical inclusive or			Flags O D I T S Z A P C 0 X X U X 0
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, register	3	—	2	OR AL, BL
register, memory	9+EA	1	2-4	OR DX, PORT ID [DI]
memory, register	16+EA	2	2-4	OR FLAG BYTE, CL
accumulator, immediate	4	—	2-3	OR AL, 01101100B
register, immediate	4	—	3-4	OR CX,01H
memory, immediate	17+EA	2	3-6	OR [BX].CMD WORD,0CFH

OUT	**OUT** port,accumulator Output byte or word			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
immed8, accumulator	10	1	2	OUT 44, AX
DX, accumulator	8	1	1	OUT DX, AL

POP	**POP** destination Pop word off stack			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register	8	1	1	POP DX
seg-reg (CS illegal)	8	1	1	POP DS
memory	17+EA	2	2-4	POP PARAMETER

POPF	**POPF** (no operands) Pop flags off stack			Flags O D I T S Z A P C R R R R R R R R R
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	8	1	1	POPF

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

PUSH

PUSH source
Push word onto stack

Flags: O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
register	11	1	1	PUSH SI
seg-reg (CS legal)	10	1	1	PUSH ES
memory	16+EA	2	2-4	PUSH RETURN_CODE [SI]

PUSHF

PUSHF (no operands)
Push flags onto stack

Flags: O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	10	1	1	PUSHF

RCL

RCL destination,count
Rotate left through carry

Flags: O D I T S Z A P C — O: X, C: X

Operands	Clocks	Transfers*	Bytes	Coding Example
register, 1	2	—	2	RCL CX, 1
register, CL	8+4/bit	—	2	RCL AL, CL
memory, 1	15+EA	2	2-4	RCL ALPHA, 1
memory, CL	20+EA+ 4/bit	2	2-4	RCL [BP].PARM, CL

RCR

RCR designation,count
Rotate right through carry

Flags: O D I T S Z A P C — O: X, C: X

Operands	Clocks	Transfers*	Bytes	Coding Example
register, 1	2	—	2	RCR BX, 1
register, CL	8+4/bit	—	2	RCR BL, CL
memory, 1	15+EA	2	2-4	RCR [BX].STATUS, 1
memory, CL	20+EA+ 4/bit	2	2-4	RCR ARRAY [DI], CL

REP

REP (no operands)
Repeat string operation

Flags: O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	REP MOVS DEST, SRCE

REPE/REPZ

REPE/REPZ (no operands)
Repeat string operation while equal/while zero

Flags: O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	REPE CMPS DATA, KEY

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

REPNE/REPNZ

REPNE/REPNZ (no operands)
Repeat string operation while not equal/not zero

Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	REPNE SCAS INPUT LINE

RET

RET optional-pop-value
Return from procedure

Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Example
(intra-segment, no pop)	8	1	1	RET
(intra-segment, pop)	12	1	3	RET 4
(inter-segment, no pop)	18	2	1	RET
(inter-segment, pop)	17	2	3	RET 2

ROL

ROL destination,count
Rotate left

Flags	O	D	I	T	S	Z	A	P	C
	X								X

Operands	Clocks	Transfers	Bytes	Coding Examples
register, 1	2	—	2	ROL BX, 1
register, CL	8+4/bit	—	2	ROL DI, CL
memory, 1	15+EA	2	2-4	ROL FLAG BYTE [DI],1
memory, CL	20+EA+ 4/bit	2	2-4	ROL ALPHA ,CL

ROR

ROR destination,count
Rotate right

Flags	O	D	I	T	S	Z	A	P	C
	X								X

Operand	Clocks	Transfers*	Bytes	Coding Example
register, 1	2	—	2	ROR AL, 1
register, CL	8+4/bit	—	2	ROR BX, CL
memory, 1	15+EA	2	2-4	ROR PORT STATUS, 1
memory, CL	20+EA+ 4/bit	2	2-4	ROR CMD WORD, CL

SAHF

SAHF (no operands)
Store AH into flags

Flags	O	D	I	T	S	Z	A	P	C
					R	R	R	R	R

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	4	—	1	SAHF

SAL/SHL

SAL/SHL destination,count
Shift arithmetic left/Shift logical left

Flags	O	D	I	T	S	Z	A	P	C
	X								X

Operands	Clocks	Transfers*	Bytes	Coding Examples
register,1	2	—	2	SAL AL,1
register, CL	8+4/bit	—	2	SHL DI, CL
memory,1	15+EA	2	2-4	SHL [BX].OVERDRAW, 1
memory, CL	20+EA+ 4/bit	2	2-4	SAL STORE COUNT, CL

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

SAR

SAR destination,source
Shift arithmetic right

Flags: O D I T S Z A P C

O	D	I	T	S	Z	A	P	C
X				X	X	U	X	X

Operands	Clocks	Transfers*	Bytes	Coding Example
register, 1	2	—	2	SAR DX, 1
register, CL	8+4/bit	—	2	SAR DI, CL
memory, 1	15+EA	2	2-4	SAR N BLOCKS, 1
memory, CL	20+EA+ 4/bit	2	2-4	SAR N BLOCKS, CL

SBB

SBB destination,source
Subtract with borrow

O	D	I	T	S	Z	A	P	C
X				X	X	X	X	X

Operands	Clocks	Transfers*	Bytes	Coding Example
register, register	3	—	2	SBB BX, CX
register, memory	9+EA	1	2-4	SBB DI, [BX].PAYMENT
memory, register	16+EA	2	2-4	SBB BALANCE, AX
accumulator, immediate	4	—	2-3	SBB AX, 2
register, immediate	4	—	3-4	SBB CL, 1
memory, immediate	17+EA	2	3-6	SBB COUNT [SI], 10

SCAS

SCAS dest-string
Scan string

O	D	I	T	S	Z	A	P	C
X				X	X	X	X	X

Operands	Clocks	Transfers*	Bytes	Coding Example
dest-string	15	1	1	SCAS INPUT LINE
(repeat) dest-string	9+15/rep	1/rep	1	REPNE SCAS BUFFER

SEGMENT†

SEGMENT override prefix
Override to specified segment

O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	MOV SS:PARAMETER, AX

SHR

SHR destination,count
Shift logical right

O	D	I	T	S	Z	A	P	C
X								X

Operands	Clocks	Transfers*	Bytes	Coding Example
register, 1	2	—	2	SHR SI, 1
register, CL	8+4/bit	—	2	SHR SI, CL
memory, 1	15+EA	2	2-4	SHR ID BYTE [SI][BX], 1
memory, CL	20+EA+ 4/bit	2	2-4	SHR INPUT WORD, CL

SINGLE STEP†

SINGLE STEP (Trap flag interrupt)
Interrupt if TF = 1

O	D	I	T	S	Z	A	P	C
		0	0					

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	50	5	N/A	N/A

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

†ASM-86 incorporates the segment override prefix into the operand specification and not as a separate instruction. SEGMENT is included only for timing information.

†SINGLE STEP is not an instruction, it is included only for timing information.

STC

STC (no operands)
Set carry flag

Flags	O	D	I	T	S	Z	A	P	C
									1

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	STC

STD

STD (no operands)
Set direction flag

Flags	O	D	I	T	S	Z	A	P	C
		1							

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	STD

STI

STI (no operands)
Set interrupt enable flag

Flags	O	D	I	T	S	Z	A	P	C
			1						

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	STI

STOS

STOS dest-string
Store byte or word string

Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Example
dest-string	11	1	1	STOS PRINT_LINE
(repeat) dest-string	9+10/rep	1/rep	1	REP STOS DISPLAY

SUB

SUB destination,source
Subtraction

Flags	O	D	I	T	S	Z	A	P	C
	X				X	X	X	X	X

Operands	Clocks	Transfers*	Bytes	Coding Example
register, register	3	—	2	SUB CX, BX
register, memory	9+EA	1	2-4	SUB DX, MATH_TOTAL [SI]
memory, register	16+EA	2	2-4	SUB [BP+2], CL
accumulator, immediate	4	—	2-3	SUB AL, 10
register, immediate	4	—	3-4	SUB SI, 5280
memory, immediate	17+EA	2	3-6	SUB [BP].BALANCE, 1000

TEST

TEST destination,source
Test or non-destructive logical and

Flags	O	D	I	T	S	Z	A	P	C
	0				X	X	U	X	0

Operands	Clocks	Transfers*	Bytes	Coding Example
register, register	3	—	2	TEST SI, DI
register, memory	9+EA	1	2-4	TEST SI, END_COUNT
accumulator, immediate	4	—	2-3	TEST AL, 00100000B
register, immediate	5	—	3-4	TEST BX, 0CC4H
memory, immediate	11+EA	—	3-6	TEST RETURN_CODE, 01H

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

WAIT	**WAIT** (no operands) Wait while $\overline{\text{TEST}}$ pin not asserted			**Flags** O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	3 + 5n	—	1	WAIT

XCHG	**XCHG** destination,source Exchange			**Flags** O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
accumulator, reg16	3	—	1	XCHG AX, BX
memory, register	17+EA	2	2-4	XCHG SEMAPHORE, AX
register, register	4	—	2	XCHG AL, BL

XLAT	**XLAT** source-table Translate			**Flags** O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
source-table	11	1	1	XLAT ASCII_TAB

XOR	**XOR** destination,source Logical exclusive or			**Flags** O D I T S Z A P C 0 X X U X 0
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, register	3	—	2	XOR CX, BX
register, memory	9+EA	1	2-4	XOR CL, MASK_BYTE
memory, register	16+EA	2	2-4	XOR ALPHA [SI], DX
accumulator, immediate	4	—	2-3	XOR AL, 01000010B
register, immediate	4	—	3-4	XOR SI, 00C2H
memory, immediate	17+EA	2	3-6	XOR RETURN_CODE, 0D2H

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

8086 - 8088
PROGRAMMAZIONE

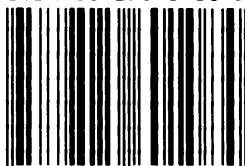
JAMES W. COFFRON

Questo testo, che consente di utilizzare al meglio i microprocessori 8086 e 8088 sfruttandone appieno le effettive potenzialità, è suddiviso in due sezioni.
Nella prima, che permette di approfondire i concetti generali di programmazione, sono sviluppati i fondamenti dei due microprocessori e della loro struttura interna, l'organizzazione della memoria, la gestione delle interruzioni e le tecniche di input/output.
Viene anche fornito l'elenco completo delle istruzioni.
La seconda, che affronta le applicazioni e l'utilizzo reale del microprocessore, descrive la famiglia dei PC-IBM e la loro architettura hardware, il BIOS (basic input/output system), il PC-DOS e l'MS-DOS. Vengono trattati, inoltre, i file di tipo COM e EXE e gli strumenti di sviluppo in ambiente DOS.
Il testo è corredato da una serie di esercizi riepilogativi che consentono di verificare e di chiarire i concetti descritti.
Le istruzioni disponibili sono facilmente utilizzabili e permettono di costruire programmi chiari e semplici.
Le differenze e le caratteristiche comuni dei due microprocessori 8086 e 8088 sono trattate e discusse in ogni particolare.
Lo studio di questo volume sarà utile sia all'esperto di programmazione, sia al programmatore alle prime armi con il linguaggio assembly.

- Concetti di programmazione
- Struttura interna dell'8086/8088
- Set completo di istruzioni
- Tecniche di interruzione, tecniche di input e di output
- Architettura hardware del PC IBM
- BIOS e DOS
- Sviluppo dei programmi sotto DOS

GRUPPO EDITORIALE JACKSON

ISBN 88-256-0108-5

L. 49.000 Cod. BY878

817

JAMES W. COFFRON

8086 - 8088 PROGRAMMAZIONE